GRAU EN ENGINYERIA INFORMÀTICA

ESPECIALITAT EN TECNOLOGIES DE LA INFORMACIÓ

Open Overlay Router a Apple iOS

Autor:

Oriol Marí Marqués

Director:

Albert Cabellos Aparicio

(Arquitectura de Computadors)

22 de Setembre de 2019

Resum

El projecte consisteix a en la cerca de la viabilitat i el desenvolupament d'una aplicació que permeti executar *Open Overlay Router* sobre el sistema operatiu *Apple iOS*.

Open Overlay Router és un implementació de codi amb llicencia *Apache 2.0* per crear xarxes de *overlay* programables, esta disponible per als sistemes operatius *Android, Linux i OpenWrt.*

Amb aquest projecte incorporarem *Apple iOS* a la llista de sistemes operatius suportats.

Resumen

El proyecto consiste en la investigación de la viabilidad y el desarrollo de una aplicación que permita ejecutar *Open Overlay Router* sobre el sistema operativo *Apple iOS*.

Open Overlay Router es una implementación de código con licencia *Apache 2.0* que permite crear redes *Overlay* programables, esta disponible para los sistemas operativos *Android, Linux* y *OpenWRT*.

Con este proyecto incorporaremos *Apple iOS* a la lista de sistemas operativos soportados.

Abstract

This project consist of the feasibility investigation and the development of an application that allows to run Open Overlay Router on the *Apple iOS* operating System.

Open Overlay Router is an implementation of code with *Apache 2.0* license that allows you to create programable overlay networks. Is available for *Android, Linux* and *OpenWRT* operating Systems.

With this project we will incorporate *Apple iOS* to the list of supported operating systems.

Índex

1. INTRODUCCIÓ ..... - 9 -

1.1. FORMULACIÓ DEL PROBLEMA ..... - 9 -

2. CONTEXT ..... - 11 -

2.1. ACTORS IMPLICATS ..... - 11 -

2.2. ESTAT DE L'ART ..... - 11 -

2.3. ABAST DEL PROJECTE ..... - 12 -

2.3.1. Objectius ..... - 12 -

2.3.2. Possibles obstacles ..... - 13 -

2.4. METODOLOGIA ..... - 14 -

3. PLANIFICACIÓ TEMPORAL ..... - 15 -

3.1. DESCRIPCIÓ DE LES TASQUES ..... - 15 -

3.1.1. Anàlisis de l'aplicació existent ..... - 15 -

3.1.2. Familiarització amb el llenguatge de programació Swift ..... - 15 -

3.1.3. Cerca de viabilitats ..... - 16 -

3.1.4. Desenvolupament de l'aplicació ..... - 16 -

3.1.5. Publicació de l'aplicació a App Store d'Apple ..... - 16 -

3.1.6. Documentació i preparació de la defensa ..... - 16 -

3.2. ESTIMACIÓ DEL TEMPS I SEQÜENCIA DE TASQUES ..... - 17 -

3.2.1. Temps estimat per tasca ..... - 17 -

3.2.2. Diagrama de Gantt ..... - 17 -

3.3. POSSIBLES COMPLICACIONS I ALTERNATIVES ..... - 17 -

4. GESTIÓ ECONÒMICA I SOSTENIBILITAT ..... - 19 -

4.1. IDENTIFICACIÓ DELS COSTOS ..... - 19 -

4.1.1. Recursos humans ..... - 19 -

4.1.2. Recursos hardware ..... - 19 -

4.1.3. Recursos software ..... - 20 -

4.1.4. Cost total ..... - 20 -

4.1.5. Control de gestió ..... - 20 -

4.2. SOSTENIBILITAT ..... - 21 -

4.2.1. Dimensió econòmica ..... - 21 -

4.2.2. Dimensió social ..... - 22 -

4.2.3. Dimensió ambiental ..... - 22 -

4.2.4. Taula de sostenibilitat ..... - 22 -

5. ANÀLISIS DE L'APLICACIÓ EXISTENT ..... - 23 -

5.1. LOCATOR ID SEPARATION PROTOCOL (LISP) ..... - 23 -
5.2. PORTABLE OPERATING SYSTEM INTERFACE (POSIX) ..... - 25 -
5.3. APPLE IOS ..... - 26 -
5.4. INTERFÍCIE DE XARXA VIRTUAL TUN ..... - 27 -
5.5. OPEN OVERLAY ROUTER (OOR) ..... - 28 -

6. DESENVOLUPAMENT D'APLICACIONS A APPLE IOS ..... - 30 -

6.1. APPLE DEVELOPER PROGRAM ..... - 30 -
6.2. XCODE ..... - 30 -
6.3. SWIFT ..... - 31 -
6.4. NETWORKEXTENSION FRAMEWORK ..... - 31 -
6.4.1. Packet Tunnel Provider ..... - 32 -
6.4.2. Entitlements ..... - 32 -
6.5. CAPTURA DE PAQUETS ..... - 32 -
6.5.1. Wireshark ..... - 33 -

7. DISSENY DE L'APLICACIÓ ..... - 34 -

7.1. VERSIÓ 1: INTERCANVI DE PAQUETS MITJANÇANT SOCKETS LOCALS ..... - 37 -
7.2. VERSIÓ 2: INTERCANVI DE PAQUETS MITJANÇANT BUFFERS DE MEMÒRIA ..... - 37 -

8. DESENVOLUPAMENT DE L'APLICACIÓ ..... - 39 -

8.1. COMPILACIÓ AMB LLVM ..... - 39 -
8.1.1. Libconfuse ..... - 39 -
8.1.2. Preprocessador de C i directives de compilació ..... - 40 -
8.1.3. Bridging headers ..... - 43 -
8.2. DESENVOLUPAMENT DE NOVES PARTS ..... - 44 -
8.2.1. Paraula Extern i punters a funcions ..... - 44 -
8.2.2. PacketTunnelProvider ..... - 47 -
8.2.3. Data-plane ..... - 52 -
8.2.4. Network manager (net-mgr) ..... - 54 -
8.2.5. Timer Wheel ..... - 54 -
8.3. INTERFÍCIE GRÀFICA ..... - 55 -
8.3.1. View Controllers ..... - 55 -
8.3.2. Storyboards ..... - 57 -
8.3.3. Vista principal ..... - 58 -
8.3.4. Configuració ..... - 59 -
8.3.5. Registre ..... - 60 -

9. RESULTATS ..... - 61 -

10. CONCLUSIONS ..... - 63 -

11. TREBALL FUTUR ..... - 64 -

12. REFERÈNCIES ..... - 65 -

Índex d'il·lustracions

Il·lustració 1: Una xarxa overlay comparada amb la seva xarxa física .....- 10 -
Il·lustració 2: Diagrama de Gantt amb les tasques .....- 17 -
Il·lustració 3: Diagrama basic de funcionament del protocol LISP .....- 24 -
Il·lustració 4: Capçalera LISP .....- 25 -
Il·lustració 5: Arbre de distribucions basades en el sistema operatiu UNIX..- 27 -
Il·lustració 6: Diagrama amb aplicació, App Extension i App Group .....- 35 -
Il·lustració 7: Diagrama de flux de paquets.....- 35 -
Il·lustració 8: Diagrama de intercanvi de paquets, primera versió.....- 37 -
Il·lustració 9: Diagrama de intercanvi de paquets, segona versió .....- 38 -
Il·lustració 10: Opció GNU89 a Xcode.....- 39 -
Il·lustració 11: Importació de llibreries externes a Xcode .....- 40 -
Il·lustració 12: Configuració de Bridging Header a Xcode.....- 43 -
Il·lustració 13: Diagrama de un view controller.....- 56 -
Il·lustració 14: LaunchScreen.storyboard.....- 57 -
Il·lustració 15: Main.storyboard.....- 58 -
Il·lustració 16: Vista principal.....- 58 -
Il·lustració 17: Vista de configuració .....- 60 -
Il·lustració 18: Vista de registre de successos .....- 60 -
Il·lustració 19: Captura de wireshark sense Open Overlay Router .....- 61 -
Il·lustració 20: Captura de wireshark amb Open Overlay Router funcionant - 61 -
Il·lustració 21: Prova a whatsmyip.com .....- 62 -

Índex de taules

Taula 1: Resum de hores per tasca .....- 17 -
Taula 2: Costos de recursos humans.....- 19 -
Taula 3: Costos de recursos hardware .....- 20 -
Taula 4: Costos de recursos software .....- 20 -
Taula 5: Cost total.....- 20 -
Taula 6: Puntuació ambiental .....- 22 -

Índex de fragments de codi

Codi 1: Directives de preprocessador C endians.h ..... - 41 -
Codi 2: Directives de preprocessador de C amb estructures IP ..... - 41 -
Codi 3: Directives de preprocessador de C amb logs ..... - 42 -
Codi 4: Directives de preprocessador de C amb la llibreria Netlink ..... - 42 -
Codi 5: Bridging Header ..... - 43 -
Codi 6: Estructura control_dplane_struct ..... - 45 -
Codi 7: Funció control_dp_select() ..... - 46 -
Codi 8: Estructura control_dp_apple ..... - 46 -
Codi 9: Funció startTunnel() ..... - 47 -
Codi 10: Funció startOOR() ..... - 48 -
Codi 11: Funció startTunnel(), versió 2 ..... - 49 -
Codi 12: Funció handlePackets(), versió 1 ..... - 50 -
Codi 13: Funció handlePackets(), versió 2 ..... - 50 -
Codi 14: Funció reachabilityChanged() ..... - 51 -
Codi 15: Funció ios_init() ..... - 52 -
Codi 16: Funció ios_output_recv() ..... - 52 -
Codi 17: Funció ios_process_input_packet() ..... - 53 -
Codi 18: Funció ios_output_recv2() ..... - 53 -
Codi 19: Funció ios_process_input_packet() ..... - 54 -
Codi 20: Funció ios_netm_init() ..... - 54 -
Codi 21: Estructura timer_wheel ..... - 55 -

1. Introducció

Aquest projecte és un projecte de Treball de Fi de Grau del Grau en Enginyeria Informàtica de la Facultat d'Informàtica de Barcelona, i desenvolupat en el marc d'un projecte de col·laboració entre el Departament d'Arquitectura de Computadors[1] i Cisco Systems, Inc[2], anomenat *Open Overlay Router*[3] (OOR).

En aquest projecte s'analitza la viabilitat i es desenvolupa una versió de *Open Overlay Router* per al sistema operatiu *Apple iOS*[4] que utilitzen els dispositius *iPhone*[5] i *iPad*[6] del fabricant Apple[7].

1.1. Formulació del problema

Amb l'avanç de les tecnologies i la massificació del us de Internet, s'ha requerit del desenvolupament de noves eines que s'ajustin a les necessitats actuals. Tradicionalment, l'enrutament a internet ha utilitzat un únic espai de noms per expressar simultàniament la identitat i al ubicació de un dispositiu dintre d'una xarxa, l'adreça IP.

El problema d'aquest tipus d'enrutament és la deficient escalabilitat, a causa de la creixent quantitat de dispositius que es connecten a Internet, les taules d'enrutament creixen de forma descontrolada creant així escassetat d'adreces IPv4[8] per a identificar tants dispositius.

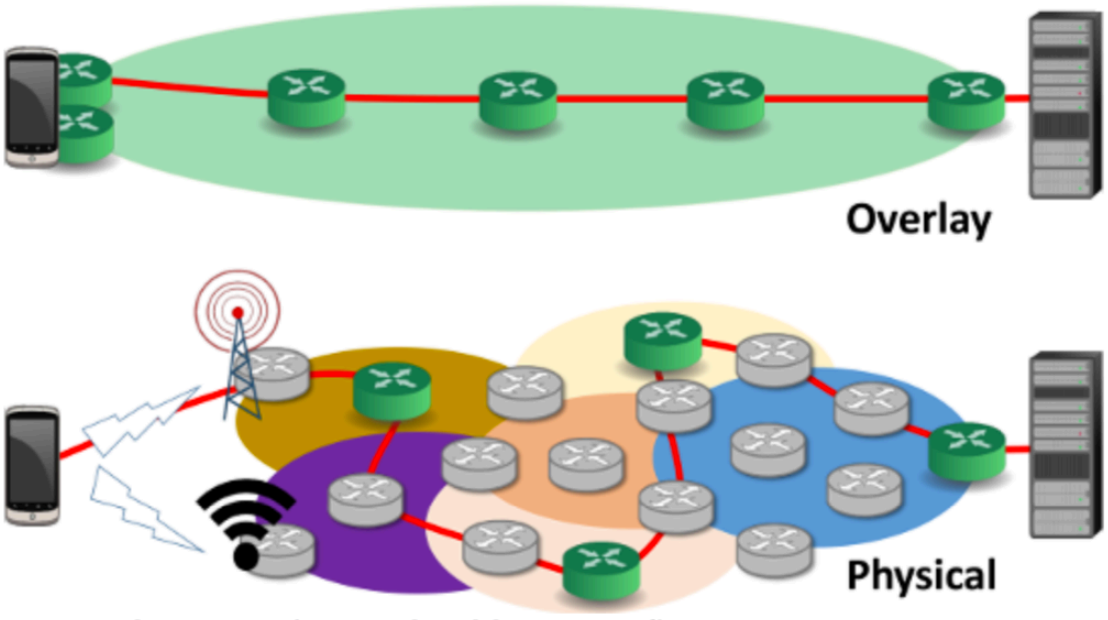
Per a intentar solucionar aquest problema, paral·lelament a IPv6[9], Cisco Systems, Inc, ha creat el protocol *Locator/Identifier Separation Protocol*[10] (LISP) per tal de millorar la velocitat i la eficiència de les taules d'enrutament. *LISP* divideix

[1] https://www.ac.upc.edu/
[2] http://cisco.com
[3] https://openoverlayrouter.org
[4] https://en.wikipedia.org/wiki/IOS
[5] https://en.wikipedia.org/wiki/IPhone
[6] https://en.wikipedia.org/wiki/IPad
[7] https://www.apple.com
[8] https://en.wikipedia.org/wiki/IPv4
[9] https://en.wikipedia.org/wiki/IPv6
[10] https://en.wikipedia.org/wiki/Locator/Identifier_Separation_Protocol

semànticament l'espai d'adreces IP en dos parts, la identitat del dispositiu o *endpoint identifier* (EID) i la ubicació o *routing locator* (RLOC).

Mitjançant la utilització de *LISP* s'aconsegueix reduir la mida de les taules d'enrutament al disminuir el numero de prefixes d'enrutament globals.

L'objectiu de Open Overlay Router és oferir una implementació de codi obert de *LISP* compatible amb els sistemes operatius *Linux*[11], *Android*[12]i *OpenWrt*[13]. D'aquesta forma els usuaris d'aquests sistemes operatius poden crear de forma fàcil xarxes *overlay*[14] de forma fàcil.



Il·lustració 1: Una xarxa overlay comparada amb la seva xarxa física

[11] https://en.wikipedia.org/wiki/Linux
[12] https://en.wikipedia.org/wiki/Android_(operating_system)
[13] https://en.wikipedia.org/wiki/OpenWrt
[14] https://en.wikipedia.org/wiki/Overlay_network

2. Context

2.1. Actors implicats

El projecte va dirigit a qualsevol usuari d'*Apple iOS* que vulgui utilitzar *Open Overlay Router* i així facilitar l'adopció del protocol *LISP* entre usuaris finals.

- Desenvolupador principal: L'autor d'aquest projecte serà la única persona que desenvolupi el projecte i encarregarà d'assolir totes les fites programades, documentar el projecte i finalment presentar-lo.

- Director de projecte: El director d'aquest projecte és Alberto Cabellos Aparicio, el paper del qual ha sigut supervisar el compliment del calendari i objectius establerts per al desenvolupament del projecte.

- Consultor: Albert López Brescó ha estat el desenvolupador del projecte *Open Overlay Router* i ha actuat com a consultor i coordinador al llarg del projecte quan aquests ho ha requerit.

- Equip de *Open Overlay Router*: Tot el grup d'enginyers involucrats en el projecte *Open Overlay Router*, incloent a l'empresa Cisco Systems, Inc, es veuran beneficiats de l'èxit del projecte ja que podran provar l'aplicació en un altre sistema operatiu diferent i ampliaran el ventall de usuaris.

- Usuaris finals: Com que és un projecte de codi obert, és gratuït i qualsevol usuari en podrà fer us i, per tant, accedir a una xarxa *LISP* amb el seu dispositiu *Apple iOS*.

2.2. Estat de l'art

A dia d'avui, s'estan desenvolupant varis projectes sobre *LISP* i cada cop hi ha més empreses que estan adoptant el seu us. Tot i això, després de fer recerca, ens trobem en que no hi ha cap aplicació que proporcioni les funcionalitats de *LISP* ni funcionalitats

similars (com ara, multihoming, IPv6 transition, etc...) al sistema operatiu *Apple iOS*. Per tant, *Open Overlay Router* serà la primera aplicació que porti el protocol *LISP*, les seves funcionalitats i beneficis a aquest sistema operatiu.

A dia d'avui no hi ha cap solució similar comparable per al sistema operatiu *Apple iOS*. Alguns exemples d'aplicacions que es podrien ubicar en la mateixa categoria, però, com mencionat anteriorment, no aportarien les mateixes funcionalitats serien *OpenVPN*[15] i *Cisco AnyConnect*[16].

2.3. Abast del projecte

2.3.1. Objectius

L'objectiu d'aquest projecte és la recerca de la viabilitat i en com es podria implementar *Open Overlay Router* al sistema operatiu *Apple iOS* i així poder utilitzar el dispositiu com a un node mòbil *LISP*, encapsulant els paquets sortints del telèfon en una capçalera *LISP* i eliminant-la dels paquets entrants. D'aquest mode els usuaris d'aquest sistema operatiu és podran beneficiar de les característiques que aporta el protocol *LISP*. Els avantatges més rellevants son:

- Multihoming: Això, en el cas dels telèfons mòbils, permet la possibilitat de utilitzar una xarxa *WiFi*[17]conjuntament amb una xarxa *LTE*[18] de forma transparent per l'usuari, habilitant així funcions com el balanceig de carrega i alta disponibilitat.

- Dynamic VPN[19] provisioning: *Open Overlay Router* permetrà als usuaris estar connectats simultàniament a diferents ubicacions remotes utilitzant diferents protocols d'encapsulament.

[15] https://en.wikipedia.org/wiki/OpenVPN
[16] https://en.wikipedia.org/wiki/Cisco_Systems_VPN_Client
[17] https://en.wikipedia.org/wiki/Wi-Fi
[18] https://en.wikipedia.org/wiki/LTE_(telecommunication)
[19] https://en.wikipedia.org/wiki/Virtual_private_network

- IPv6 transition[20]: Permetrà que els usuaris puguin accedir a xarxes IPv6 tot i estar connectats a una xarxa IPv4. Això es molt útil per ajudar al procés de migració de IPv4 a IPv6 de empreses.

Un cop trobat una forma viable de fer-ho, es procedirà a la implementació del mateix proporcionant així les funcions de *Open Overlay Router* amb suport per a adreçament IPv4 i IPv6.

Es farà una interfície gràfica des d'on configurar les propietats i variables requerides per el protocol *LISP*. A més a més l'usuari també podrà mirar els registres de l'aplicació des d'aquesta interfície.

Finalment, s'haurà de procedir a la publicació de l'aplicació a *l'App Store*[21] de Apple, els quals tenen una sèrie de requisits i filtres que s'hauran de superar i gestionar.

2.3.2. Possibles obstacles

Durant el projecte poden sorgir certes complicacions i dificultats que afectin al seu desenvolupament i modifiquin la planificació temporal, alguns exemples:

- Desconeixement del llenguatge de programació *Swift*[22].

- Possible falta d'una interfície per parlar amb el *kernel* de xarxa de *Apple iOS*.

- Apple és molt estricte en quan a l'accés de certes funcions del seu software, per tant és probable que ens trobem en limitacions a les que haguem de trobar solucions.

- Integració de codi font escrit en C de *Open Overlay Router* amb el nou codi que s'escriurà amb el llenguatge *Swift*.

[20] https://en.wikipedia.org/wiki/IPv6_transition_mechanism
[21] https://en.wikipedia.org/wiki/Apple_Store
[22] https://en.wikipedia.org/wiki/Swift_(programming_language)

- Normes i regulacions estrictes de publicació de l'aplicació a l'App Store.

2.4. Metodologia

S'ha escollit utilitzar la metodologia àgil basada en *Scrum*[23], que ens permetrà gestionar el projecte de forma setmanal, en la qual hi haurà una reunió per analitzar el compliment dels objectius i establir els del següent període. A més a més aquesta metodologia es adaptable a canvis i imprevistos.

S'utilitzarà *Github*[24] per a la gestió i emmagatzematge de codi per així tindre control de versions i gestionar els canvis i modificacions.

Per al desenvolupament de l'aplicació s'utilitzarà *Xcode*[25] com a entorn de desenvolupament integrat[26] (IDE) ja que proporciona totes les integracions i eines necessàries per desenvolupar aplicacions per *Apple iOS*.

[23] https://en.wikipedia.org/wiki/Scrum_(software_development)
[24] https://github.com
[25] https://en.wikipedia.org/wiki/Xcode
[26] https://en.wikipedia.org/wiki/Integrated_development_environment

3. Planificació temporal

El projecte te una duració d'unes 18 setmanes, començant a comptar la setmana del 9 de Setembre i esta pensat per a realitzar-se en 450 hores. En aquestes setmanes també estan incloses les tasques a realitzar per l'assignatura Gestió de Projectes (GEP), realitzada durant el primer mes del projecte.

L'objectiu d'aquesta assignatura és ajudar a encaminar el projecte i començar a documentar-lo des del principi, per tal que al memòria sigui complerta i la seva defensa posterior es realitzi correctament.

3.1. Descripció de les tasques

3.1.1. Anàlisis de l'aplicació existent

A l'inici del projecte es indispensable analitzar l'aplicació actual del sistema operatiu Linux per entendre be el funcionament de la mateixa i el codi font.

A partir d'aquest anàlisis s'extrauran les parts de codi C reutilitzables i es definiran quines funcionalitats s'hauran de reescriure en Swift. Un cop haguem entès com funciona l'aplicació actual podrem procedir amb les següents tasques.

3.1.2. Familiarització amb el llenguatge de programació Swift

Ningun membre del grup de recerca de *Open Overlay Router* ha programat mai en el llenguatge *Swift*, per tant s'hauran de dedicar un conjunt d'hores a l'autoaprenentatge d'aquest llenguatge i cercar quines interfícies i llibreries es poden utilitzar per tal de dur a terme les parts de codi necessàries per integrar amb el codi existent de *Open Overlay Router.*

Aquesta tasca és imprescindible per assegurar un bon resultat i un desenvolupament de qualitat de l'aplicació. Com més aprenguem el llenguatge més opcions i viabilitats tindrem per a poder analitzar i valorar en la següent tasca.

3.1.3. Cerca de viabilitats

Un cop familiaritzats amb el llenguatge *Swift* i sabent quines interfícies ens proporciona Apple per a desenvolupar l'aplicació, procedirem a fer proves de concepte per veure de quina forma, quines possibilitats i amb quines arquitectures podem desenvolupar l'aplicació i realitzar la integració amb el codi C. Aquesta serà una de les tasques que requerirà de més temps.

3.1.4. Desenvolupament de l'aplicació

Un cop tinguem clar el camí a seguir després de finalitzar la tasca anterior, s'hauran de desenvolupar totes les parts de codi necessàries i testejar-les. A més a més, aquesta tasca també contempla el desenvolupament de la interfície gràfica.

3.1.5. Publicació de l'aplicació a App Store d'Apple

Aquesta tasca es realitzarà de forma conjunta amb les anteriors, consisteix en entendre els requisits d'Apple i aprendre com funcionen els procediments per tal de poder publicar aplicacions a la seva botiga. Probablement ens requerirà de fer canvis a l'aplicació. En aquest procés no tenim dependències de tasques anteriors i podrem anar realitzant-la en paral·lel.

3.1.6. Documentació i preparació de la defensa

Finalment, el projecte s'ha de documentar i defensar davant d'un tribunal de professors i doctors de la facultat. La redacció del document es realitzarà un cop finalitzat el desenvolupament de l'aplicació.

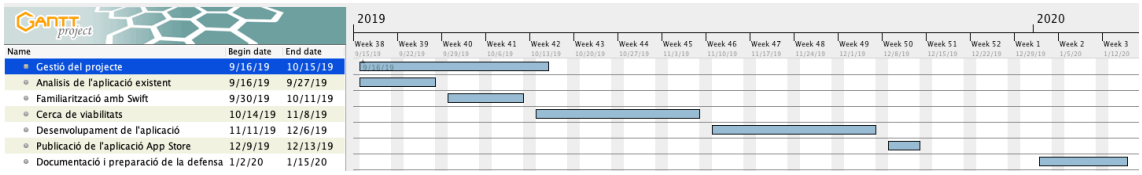
3.2. Estimació del temps i seqüencia de tasques

3.2.1. Temps estimat per tasca

Tasca	Hores estimades
Gestió del projecte	75
Anàlisis de l'aplicació existent	50
Familiarització amb llenguatge de programació *Swift*	50
Cerca de viabilitats	100
Desenvolupament de l'aplicació	100
Publicació de l'aplicació al App Store de Apple	25
Documentació i preparació de la defensa	50
Total	450

Taula 1: Resum de hores per tasca

3.2.2. Diagrama de Gantt



Il·lustració 2: Diagrama de Gantt amb les tasques

3.3. Possibles complicacions i alternatives

Tot i les estimacions i previsions per les diferents tasques, com hem mencionat anteriorment hi ha una sèrie d'obstacles que poden produir imprevistos i modificar la línia de temps del projecte.

Gracies a la metodologia utilitzada, *Scrum*, tenim fites setmanals. Això permet que en cas de que els objectius d'una setmana no es complissin, es podrien traslladar a la setmana següent o vici-versa.

De les complicacions nombrades anteriorment, tenim com a possibles solucions:

- En comptes de programar en *Swift*, programar en *Objective-C*[27].

- La falta de la interfície per parlar amb el *kernel* de xarxa de *Apple iOS* implicaria que hauríem de demanar permisos a Apple per poder fer una aplicació de més baix nivell.

- En quan a la integració dels dos llenguatges, si no es pot fer directe es pot fer mitjançant una espècie de enllaç amb un tercer llenguatge al mig, per exemple, *Objective-C*.

- El filtratge i requisits de *l'App Store*, en cas de no satisfer-los, podem publicar l'aplicació en mode "test" a un conjunt limitat d'usuaris.

[27] https://en.wikipedia.org/wiki/Objective-C

4. Gestió econòmica i sostenibilitat

4.1. Identificació dels costos

A continuació s'exposen els diferents costos aproximats relatius a cada element per dur a terme el projecte, és a dir, recursos humans, hardware i software del projecte.

4.1.1. Recursos humans

La remuneració s'ha calculat en funció de la web *LinkedIn Salary*[28] consultat a l'Octubre de 2019:

Rol	Hores	Salari per hora	Total
Enginyer de software (sense titulació)	450	10	4.500€
Total	450		4.500€

Taula 2: Costos de recursos humans

4.1.2. Recursos hardware

Els recursos hardware utilitzats s'estima que tindran una vida útil de 4 anys. En el cost d'amortització per hora es consideren els dies hàbils de l'any, que son uns 250 i 8 hores al dia.

És a dir, la formula utilitzada per calcular l'amortització és:

Cost de l'equip / (4 anys vida útil * 250 dies feiners/any * 8 hores de dedicació)

I per calcular el cost, multipliquem l'amortització per el numero de hores de dedicació de TFG (450 hores).

[28] https://www.linkedin.com/salary/

Hardware	Preu	Hores	Amortització (€/hora)	Cost
MacBook Pro	1.749€	450	0,218€	98,10€
iPhone 8	539€	450	0,067€	30,32€
Total				130€

Taula 3: Costos de recursos hardware

4.1.3. Recursos software

L'únic requisit de software que hi ha es la llicencia de desenvolupador d'Apple Enterprise necessària per poder utilitzar certes interfícies o llibreries i per a poder publicar l'aplicació a *l'App Store*.

Software	Preu
Llicencia Apple Developer Enterprise	299€ anuals

Taula 4: Costos de recursos software

4.1.4. Cost total

A continuació es mostra el resum de cost total. En aquest cost també hauríem de afegir els costos indirectes (us de locals, electricitat, internet, etc) però degut a que el projecte es realitza en espais compartits i els costos son ínfims no els afegim ja que no els considerem rellevants.

Recurs	Cost estimat
Recursos humans	4.500€
Recursos hardware	130€
Recursos software	299€
Total costos directes	4.929€
Contingències	740€
Total	5.669€

Taula 5: Cost total

4.1.5. Control de gestió

Per a les despeses que no estiguin previstes es podria augmentar el pressupost a 7.000€ per tindre marge de maniobra. A més a més a les reunions setmanals s'analitzaran els recursos invertits i si cal modificar-los.

4.2. Sostenibilitat

En aquest apartat s'analitza la sostenibilitat en tres dimensions diferents: econòmica, social i ambiental. Per a cada una d'elles s'analitzen les qüestions més rellevants que pertanyen a la fita inicial del projecte, tant per a la seva posada en producció com per a al seva vida útil. L'informa s'acompanya d'una taula on es reflexa la puntuació obtinguda segons la matriu de sostenibilitat.

No tenim experiència prèvia amb cap projecte de sostenibilitat. De les 3 dimensions amb la que ens sentim més còmode és la social, ja que som capaços de veure quin impacte tindrà socialment el projecte de forma fàcil, d'altre banda, la dimensió que ens resulta menys familiar és l'ambiental degut a que és més difícil mesurar quin impacte ambiental pot tindre una aplicació d'aquest tipus.

En quan a ergonomia i accessibilitat, en aquesta aplicació és seguiran els patrons de disseny de interfícies proposats per Apple que han demostrat complir amb els requisits de ergonomia i accessibilitat de la societat. No hem emprat mai eines per mesurar costos econòmics, ni ambientals i socials.

Disposem d'experiència en eines per treballar de forma col·laborativa en projectes TIC.

4.2.1. Dimensió econòmica

Existeix una avaluació de costos, i els tipus d'aquests. El cost del projecte es competitiu ja que la inversió estalvia temps tant a gent de l'equip com a usuaris finals i a més a més hi ha col·laboració amb el departament d'arquitectura de computadors de la UPC.

No obstant, al ser un projecte tan obert no s'ha tingut en compte el cost de reparacions o actualitzacions un cop en producció, ja que, qualsevol funcionalitat afegida requerirà de un anàlisis temporal i de costos.

4.2.2. Dimensió social

L'aplicació *Open Overlay Router* pot ser utilitzada tant per empreses com per persones particulars per a fer servir el protocol *LISP*. La inclusió del sistema operatiu *Apple iOS* al catàleg de compatibilitats de *Open* Overlay Router farà arribar el protocol i l'aplicació a tot un mercat fins ara inabordable.

4.2.3. Dimensió ambiental

Aquest projecte no te una gran repercussió ambiental ja que els únics gestos que poden afectar son els indirectes com l'electricitat o el paper. No es requereix de cap tipus de matèria primera ni es fabrica cap tipus de dispositiu, es purament software. Tot i això, cal esmentar que el hardware utilitzar es Apple i aquesta empresa disposa d'un programa de reciclatge de productes electrònics.

4.2.4. Taula de sostenibilitat

Econòmica	Social	Ambiental
9/10	8/10	7/10

Taula 6: Puntuació ambiental

5. Anàlisis de l'aplicació existent

En la primera fase del projecte l'objectiu és conèixer l'aplicació *Open Overlay Router*, estudiar el seu codi font i funcionament, detectar els seus requisits, les parts que s'hauran de modificar i les parts que s'hauran de crear de nou.

Abans de procedir amb el anàlisis, l'autor considera apropiat fer una breu descripció de algunes parts i tecnologies clau del projecte.

5.1. Locator ID Separation Protocol (LISP)

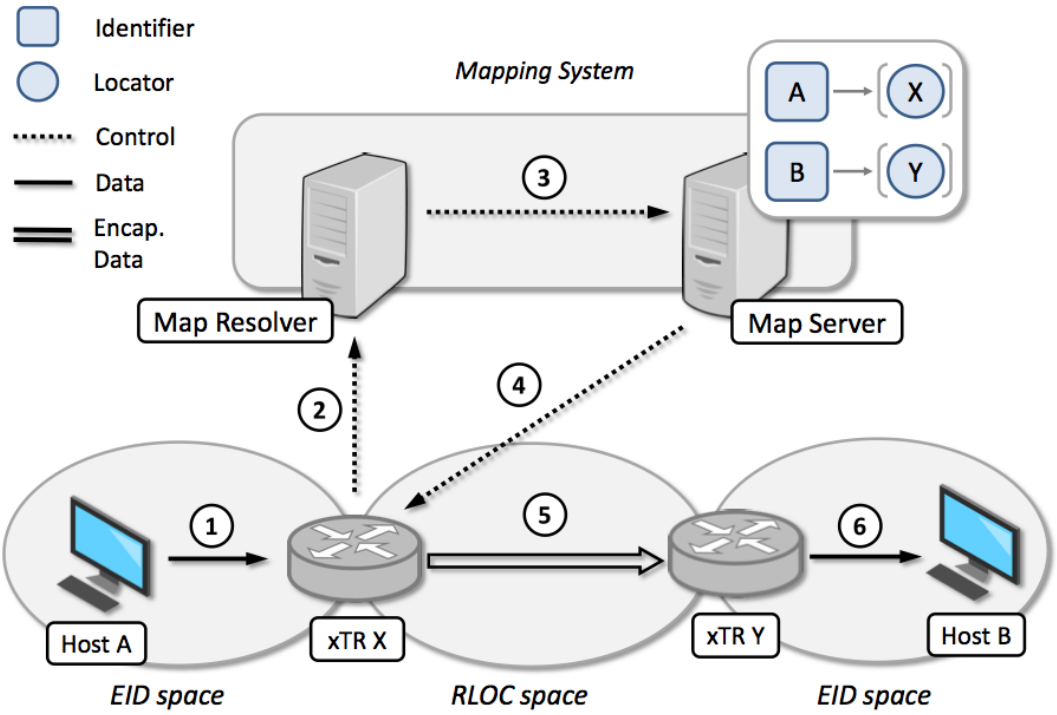
Locator ID Separation Protocol és un protocol de xarxa del tipus "map-and-encapsulate", desenvolupat per la *Internet Engineering Task Force*[29] *LISP Working Group*[30].

La idea bàsica de separació consisteix en que actualment les xarxes de computadors utilitzen l'adreça IP per indicar la identitat del dispositiu així com també la manera com esta connectat a la xarxa. Això causa que el nombre d'adreces IP disponibles sigui molt baix degut al creixement dels dispositius que utilitzen Internet, en part causat per l'èxit creixent de l'Internet de les Coses[31] (IoT).

LISP pretén solucionar el problema separant en dos espais de noms el "qui" i el "com" d'aquests dispositius connectats que utilitzen adreces IP, utilitzant un sistema de "Mapping" per relacionar les dues parts.

A continuació es troba l'estructura habitual d'un entorn de xarxa LISP:

[29] https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force
[30] https://datatracker.ietf.org/group/lisp/about/
[31] https://en.wikipedia.org/wiki/Internet_of_things



Il·lustració 3: Diagrama basic de funcionament del protocol LISP

La forma de funcionament bàsica de *LISP* consisteix en que els hosts tenen un *endpoint identifier* (EID) i estan connectats a un *Egress/Ingress tunnel router* (xTR) (un enrutador compatible amb *LISP*). Els hosts nomes es poden comunicar entre ells mitjançant els xTR.

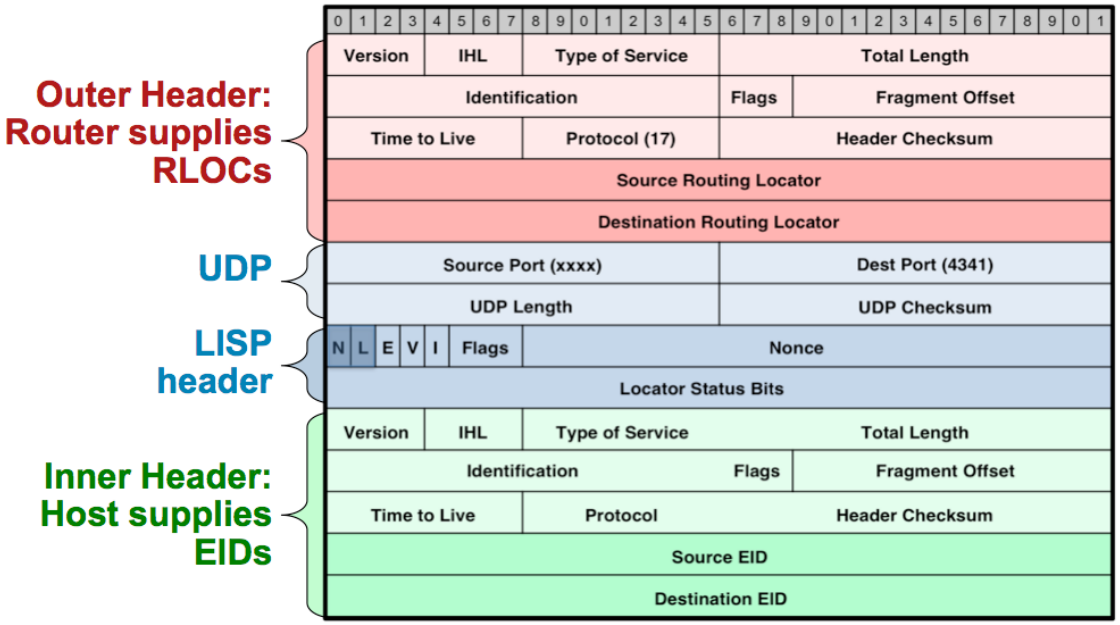
Quan el host A vol comunicar-se amb un el host B envia el paquet al xTR X i aquest pregunta al *Mapping System* a quin xTR esta connectat el host B. Un cop te la resposta el xTR X encapsula el paquet i l'envia al xTR Y, que un cop rebut, desencapsula el paquet i li reenvia al host B. Aquest procediment de encapsulació i desencapsulació és transparent tant per els hosts com per les aplicacions.

Alguns dels beneficis que aporta *LISP* son:

- Multihoming simplificat.
- Facilita l'escalabilitat en connexions WAN *any-to-any.*
- Proporciona mobilitat de maquines virtuals entre centres de dades geogràficament separats.

- Millora la escalabilitat del sistema de enrutament, ja que agrega els RLOCs.
- Optimitza l'enrutament IP tant a IPv4 com a IPv6.
- Redueix la complexitat de les operacions.

La següent imatge mostra la capçalera de un paquet enrutat amb *LISP*.



Il·lustració 4: Capçalera LISP

5.2. Portable Operating System Interface (POSIX)

POSIX[32] és una família d'estàndards de crides al sistema operatiu definits per IEEE 1003. Aquest estàndard busca generalitzar les interfícies dels sistemes operatius perquè una mateixa aplicació pugui ser executada en diferents plataformes. D'aquesta manera es simplifica els desenvolupament d'aplicacions que requereixen de un us extensiu de crides al sistema operatiu.

Aquests estàndards van sorgir d'un projecte de normalització de les API[33] i descriuen un conjunt d'interfícies d'aplicació adaptables a una gran varietat

[32] https://en.wikipedia.org/wiki/POSIX
[33] https://en.wikipedia.org/wiki/Application_programming_interface

d'implementacions dels sistemes operatius. Algunes de les crides que implementa s'utilitzen per a:

- Creació i control de processos.
- Senyals.
- Operacions de fitxers i directoris.
- Instruccions d'entrada/sortida i control de dispositius.
- Canals (Pipes).
- Planificació de processos amb prioritat (scheduling).
- Temporitzadors.
- Semàfors.

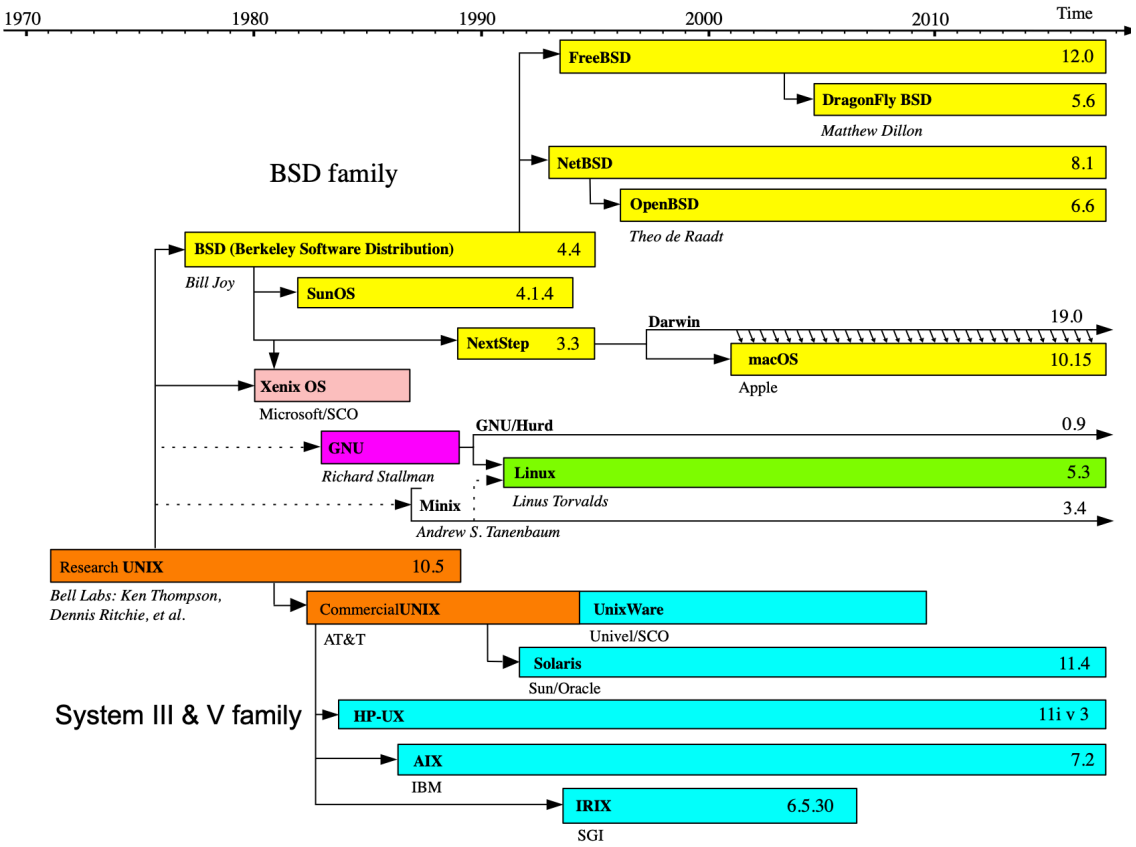
Els sistemes operatius d'avui en dia poden escollir ser compatibles al 100% amb POSIX, el que significaria que son POSIX-certified, o be poden complir amb la majoria de crides, en aquest cas serien POSIX-compliant.

5.3. Apple iOS

Es important entendre d'on ve *Apple iOS* per tal de veure d'on sorgeixen els reptes i la complexitat del projecte.

Apple iOS deriva del sistema operatiu *macOS*[34] que a la vegada aquest esta basat en el sistema operatiu *Darwin*[35]. Ens trobem amb que *Darwin* no es un sistema operatiu *Linux*, sinó que deriva de *UNIX*[36] i a més a més no es POSIX-certified tot i que si que segueix els estàndards en gran part. Veurem com aquesta condició afecta al desenvolupament de l'aplicació.

[34] https://en.wikipedia.org/wiki/MacOS
[35] https://en.wikipedia.org/wiki/Darwin_(operating_system)
[36] https://en.wikipedia.org/wiki/Unix



Il·lustració 5: Arbre de distribucions basades en el sistema operatiu UNIX

5.4. Interfície de xarxa virtual TUN

Una interfície de xarxa virtual TUN[37] es un dispositiu de xarxa software que no depèn de un adaptador de xarxa físic. Son utilitzades per tal de poder manipular els paquets de xarxa abans de que arribin a una interfície de xarxa física.

El sistema operatiu reenvia tots els paquets cap aquesta interfície i aquests son entregats a una aplicació en l'espai d'usuari que esta connectada a aquesta interfície TUN. L'aplicació també pot enviar paquets a la interfície TUN i llavors aquesta entrega els paquets al sistema operatiu emulant la seva recepció des de un origen extern.

Les interfícies TUN tenen molts casos d'us però el més conegut es per tal de crear connexions virtuals privades (VPN). *Open Overlay Router* fa us d'aquest tipus d'interfícies per tal de capturar els paquets i processar-los.

[37] https://en.wikipedia.org/wiki/TUN/TAP

5.5. Open Overlay Router (OOR)

Open Overlay Router (OOR) i pronunciat en angles *double-O R,* es una aplicació de software lliure amb el codi disponible a Github, per a crear xarxes *overlay* programables. Suporta múltiples encapsulacions, com ara *VXLAN-GPE*[38] o *LISP* i varis plans de control, com ara *netconf*[39]o models *yang*[40].

Ofereix les funcionalitats de *LISP* a altres sistemes operatius com ara *Linux*, *Android* i *OpenWrt*, ja que *LISP* de forma nativa nomes funciona sobre enrutadors amb el sistema operatiu *Cisco iOS*[41].

Open Overlay Router esta escrita amb el llenguatge de programació C i s'utilitza mitjançant la línia de comandes ja que no disposa de interfície gràfica. A dia d'avui l'aplicació a estat portada a diferents sistemes operatius per aquests sempre han estat basats en *Linux*. Aquest serà el primer cop que s'intenta portar l'aplicació a un sistema operatiu no *Linux*.

El funcionament basic de l'aplicació es similar al de una aplicació per a crear túnels VPN. Aquesta crea una interfície de xarxa virtual tipus TUN i afegeix rutes al sistema operatiu per tal d'enrutar tot el tràfic cap a aquesta interfície.

Quan el tràfic arriba a aquesta interfície, l'aplicació captura els paquets i els analitza per tal de saber quin és el destí i poder fer les consultes pertinents al sistema de mapping de la xarxa *LISP*. Un cop sap cap a on s'ha d'enviar cada paquet, afegeix la capçalera *LISP* al paquet i l'envia per la interfície de xarxa corresponent.

Observem que a la gran majoria de fitxers del codi font hi ha referencies a llibreries de *Linux* i crides a les API de POSIX, aquests fitxers s'hauran de modificar i adaptar-los al sistema operatiu *Apple iOS*. A més a més podem veure 3 components clau:

[38] https://datatracker.ietf.org/doc/draft-quinn-vxlan-gpe/
[39] https://en.wikipedia.org/wiki/NETCONF
[40] https://en.wikipedia.org/wiki/YANG
[41] https://en.wikipedia.org/wiki/Cisco_IOS

- El control-data-plane, que processa els paquets de control que s'intercanvien amb el RLOC de la xarxa LISP. En funció del contingut d'aquesta paquets de control, el control-data-plane utilitzarà el network manager per tal de aplicar modificacions a la configuració de xarxa segons sigui convenient.

- El data-plane, s'encarrega de processar, encapsular i desencapsular els paquets amb les dades de les aplicacions i d'usuari.

- El net-mgr, és l'encarregat de proporcionar la informació de xarxa de les interfícies al control-data-plane per tal de que aquest pugui prendre decisions en funció de l'estat de connectivitat de cada interfície.

Aquests components s'hauran de fer de nou, degut a que son diferents per cada sistema operatiu tot i que s'intentarà aprofitar al màxim el codi existent i que sigui el més semblant possible al dels altres sistemes operatius per tal de facilitar la incorporació de noves funcionalitats.

6. Desenvolupament d'aplicacions a Apple iOS

El desenvolupament d'aplicacions per *Apple iOS* és tot un món propietari, en el qual és pràcticament obligatori seguir unes guies de programació, uns llenguatges i utilitzar unes eines concretes imposades per l'empresa Apple.

6.1. Apple developer program

Per tal de poder desenvolupar l'aplicació requerirem associar-nos i comprar la llicencia de *Apple Developer Program*[42].

Apple permet desenvolupar aplicacions sense cost sempre i quan s'executin en simuladors i no en dispositius físics.

La idiosincràsia de l'aplicació fa que requerim de un dispositiu físic ja que en els simuladors no es possible treballar amb interfícies de xarxa.

6.2. Xcode

Xcode es un entorn de desenvolupament integrat (IDE) per al sistema operatiu *macOS* que conte un conjunt d'eines creades per Apple per tal de dur a terme el desenvolupament d'aplicacions per als sistemes operatius *macOS*, *Apple iOS*, *watchOS* i *tvOS*.

És necessari en aquest projecte utilitzar Xcode, ja que és la forma més viable de desenvolupar aplicacions per a *Apple iOS*.

[42] https://en.wikipedia.org/wiki/Apple_Developer

6.3. Swift

Swift és un llenguatge de programació multiparadigma orientat a objectes creat per Apple per al desenvolupament de programari per a *Apple iOS* i *macOS*. Ha estat dissenyat per coexistir amb *Objective-C* i per ser un llenguatge segur, de desenvolupament ràpid i concís. Utilitza el compilador *LLVM*[43].

Després de haver analitzar l'aplicació actual de *Open Overlay Router* i detectat els requisits de la mateixa es va cercar quines llibreries, frameworks i API necessitaríem per tal de poder desenvolupar la versió per a *Apple iOS*. Un gran benefici de programar amb *Swift* es que permet fer crides a llibreries escrites en C de forma nativa.

6.4. NetworkExtension Framework

Apple proporciona un framework anomenat NetworkExtension que permet custiomitzar i estendre la connectivitat de xarxa dels sistemes operatius *Apple iOS* i *macOS*. Permet desenvolupar noves funcionalitats amb accés a:

- Configuració WiFi del sistema.
- Integració amb el sistema que gestiona el hotspot[44].
- Crear i administrar configuracions VPN.
- Implementar filtratge de continguts.
- Implementar un Proxy DNS.

Com s'ha descrit anteriorment, *Open Overlay Router* fa us d'una interfície TUN per capturar el tràfic. Aquest framework ens proporciona accés per crear VPNs i per tant crea una interfície TUN.

Degut a les restriccions en el desenvolupament de aplicacions de Apple, aquesta és la única forma de accedir a les operacions de xarxa del sistema. Concretament requerim la API inclosa en aquest framework anomenada *Packet Tunnel Provider*.

[43] https://en.wikipedia.org/wiki/LLVM
[44] https://en.wikipedia.org/wiki/Hotspot_(Wi-Fi)

6.4.1. Packet Tunnel Provider

Aquesta API ens permet crear un client VPN amb un protocol personalitzat i és la que utilitzarem per tal de crear la interfície TUN per després capturar els paquets que enviarem al pla de dades de *Open Overlay Router*.

6.4.2. Entitlements

Apple és molt restrictiva en quan al desenvolupament de aplicacions. Per això per tal de poder utilitzar el framework mencionat en l'apartat anterior, es necessari sol·licitar un permís especial.

Aquesta sol·licitud consisteix a omplir un formulari i enviar-lo a Apple, al cap de unes dues setmanes s'obté la resposta i ells mateixos associen aquests permisos a la compta de Apple Developer.

6.5. Captura de paquets

Donat la idiosincràsia de l'aplicació, és necessari treballar amb interfícies de xarxa reals, i com hem dit anteriorment no és possible crear interfícies TUN al simulador de *Apple iOS*.

Per tant, és de vital importància disposar de un mecanisme per tal de poder capturar els paquets que travessen les interfícies de xarxa per tal de poder entendre i diagnosticar que esta realitzant *Open Overlay Router*.

Apple proporciona una forma de capturar els paquets que circulen per totes les interfícies de xarxa de un iPhone. Es necessari connectar el iPhone per USB a un ordinador Apple Mac i utilitzar el que s'anomena interfície Virtual Remota (RVI) de tal forma que crea una interfície virtual de xarxa en el Mac a la que es duplicarà tot el tràfic que travessa les interfícies físiques del iPhone. D'aquesta forma podem capturar els paquets mitjançant un capturador de paquets. Per a això, s'ha utilitzat una aplicació anomenada Wireshark.

6.5.1. Wireshark

Wireshark, es un analitzador de protocols i s'utilitza per a realitzar anàlisis i solucionar problemes en xarxes de comunicacions. Proporciona funcionalitats similars a les de *tcpdump*[45] però afegint una interfície gràfica i opcions de filtratge i organització de la informació.

D'aquesta forma, permet veure tot el tràfic que esta passant a traves de una interfície de xarxa en mode promiscu[46], mode en el que un ordinador captura tot el tràfic que hi circula.

Wireshark es software lliure, s'executa sobre la majoria de sistemes operatius, te una corba d'aprenentatge fàcil i es molt potent.

[45] https://en.wikipedia.org/wiki/Tcpdump
[46] https://en.wikipedia.org/wiki/Promiscuous_mode

7. Disseny de l'aplicació

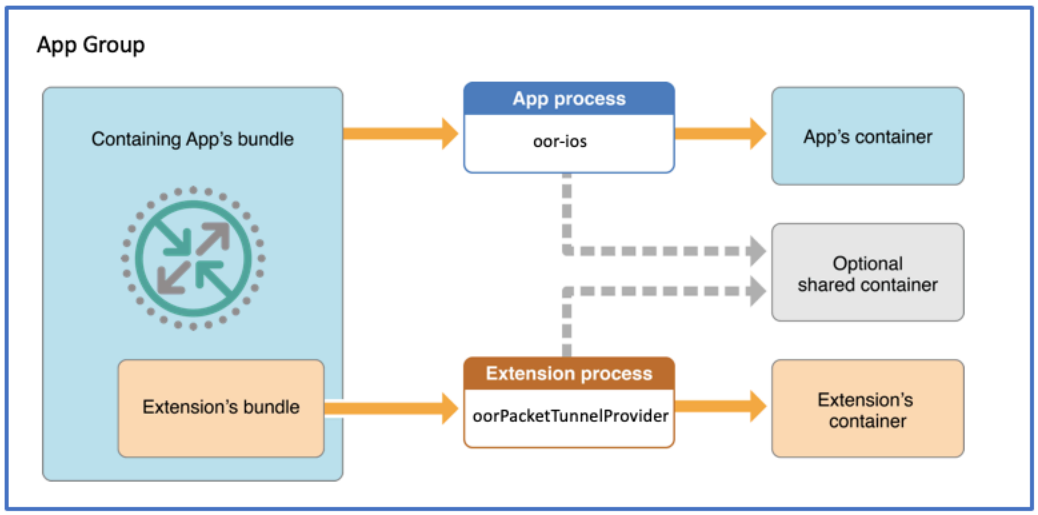
En aquesta secció es descriu a un alt nivell el disseny de l'aplicació, la qual estarà formada per dos parts, la aplicació en si que es la que mostrarà la interfície gràfica i una altre part que Apple anomena *App Extensions*[47]. Les *App extensions* s'utilitzen per proporcionar accés als usuaris a recursos fora de l'aplicació en si, ja que en el sistema operatiu *Apple iOS* totes les aplicacions s'executen en forma de contenidor dintre de un *sandbox*[48].

En el nostre cas tindrem una *App Extension* anomenada *oorPacketTunnelProvider* que serà la encarregada de gestionar els perfils VPN i la configuració de xarxa del sistema operatiu. A més a més ens permetrà continuar executant el procés de *Open Overlay Router* en segon pla, ja que en les aplicacions normals Apple limita el seu us quan no estan en primer pla.

Serà en aquesta *App Extension* on realment s'executarà el codi font de *Open Overlay Router* i on s'utilitzarà el framework de *NetworkExtension* per tal de llegir els paquets de la interfície TUN i enviar-los al *data-plane* de *Open Overlay Router.*

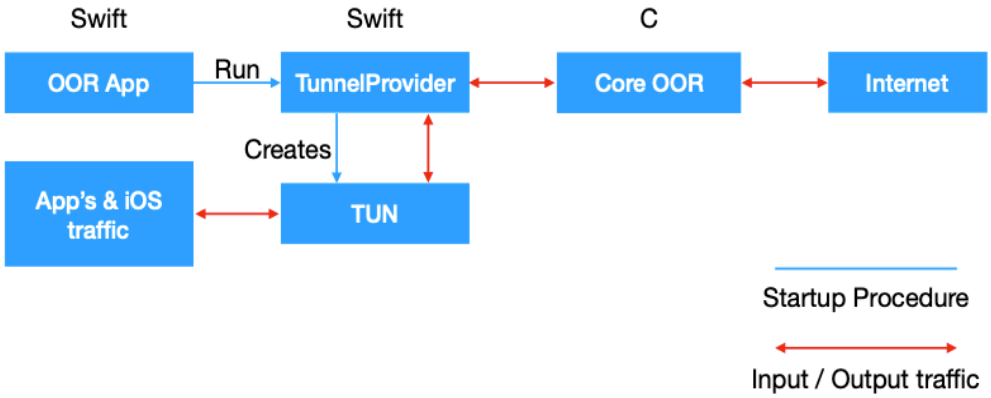
Per tal de compartir fitxers entre l'aplicació i la *App Extension* es necessari fer us del que s'anomena *App Groups*[49]. Aquesta característica permet que dues aplicacions, o en el nostre cas una *App Extension* i una aplicació obtinguin accés a recursos compartits i intercomunicació entre els processos. Concretament utilitzarem aquesta funció per poder llegir la configuració que l'usuari defineix mitjançant la interfície gràfica des de *l'App Extension.*

[47] https://developer.apple.com/app-extensions/
[48] https://en.wikipedia.org/wiki/Sandbox_(computer_security)
[49] https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_security_application-groups



Il·lustració 6: Diagrama amb aplicació, App Extension i App Group

Ara ens centrarem en el flux que seguiran els paquets que s'enviaran i es rebran entre les altres aplicacions i Internet.



Il·lustració 7: Diagrama de flux de paquets

Observem que el *TunnelProvider* es l'encarregat de crear la interfície TUN. I l'encarregat de gestionar tots paquets.

En el cas dels paquets sortints, un cop creada aquesta interfície el sistema operatiu començarà a enviar tots els paquets de altres processos cap a aquesta interfície. El *TunnelProvider* agafarà els paquets i els reenviarà cap al *data-plane* de *Open Overlay Router* el qual els analitzarà, afegirà la capçalera *LISP* i els reenviarà cap als destins corresponents de Internet.

D'altra banda, pels paquets entrants, seran rebuts per *Open Overlay Router*, que traurà la capçalera *LISP* i els reenviarà al *TunnelProvider* per a que torni a enviar-los a la interfície TUN des de on el sistema operatiu serà l'encarregat de llegir-los i tornar-los als processos corresponents.

Observem que *TunnelProvider* esta escrit en *Swift* i *Open Overlay Router* en C, i son dos processos en dos *sandbox* diferents. Aquí trobem un pas molt important a l'hora de dissenyar l'aplicació, el com s'intercanvien els paquets entre aquests dos processos.

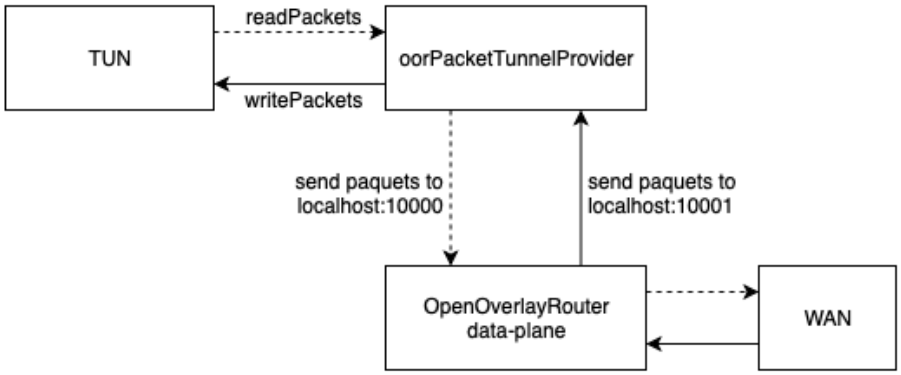
En una primera versió de l'aplicació l'objectiu principal era fer funcionar *Open Overlay Router* sense tindre en consideració el rendiment, ja que la prioritat era saber si hi havia algun mètode de fer-ho tenint en compte totes les restriccions d'Apple.

Després, un cop es va aconseguir fer funcionar l'aplicació, es va decidir millorar algunes parts, com es en aquest cas, el intercanvi de paquets entre processos a més a més d'algunes altres funcionalitats com el multihoming o que fos compatible amb IPv6.

7.1. Versió 1: Intercanvi de paquets mitjançant sockets locals

En la primera versió de l'aplicació, l'objectiu principal era fer que funciones deixant el rendiment a un costat. Amb els coneixements adquirits fins al moment es va decidir que l'intercanvi de paquets entre *TunnelProvider* i *Open Overlay Router* es realitzaria mitjançant *sockets*[50] locals.

De tal forma que, quan s'inicialitzen el *data-plane*, obrirà un port UDP i començarà a escoltar. Després, el TunnelProvider començarà a enviar els paquets que llegeixi de la interfície TUN cap a aquest port.



Il·lustració 8: Diagrama de intercanvi de paquets, primera versió

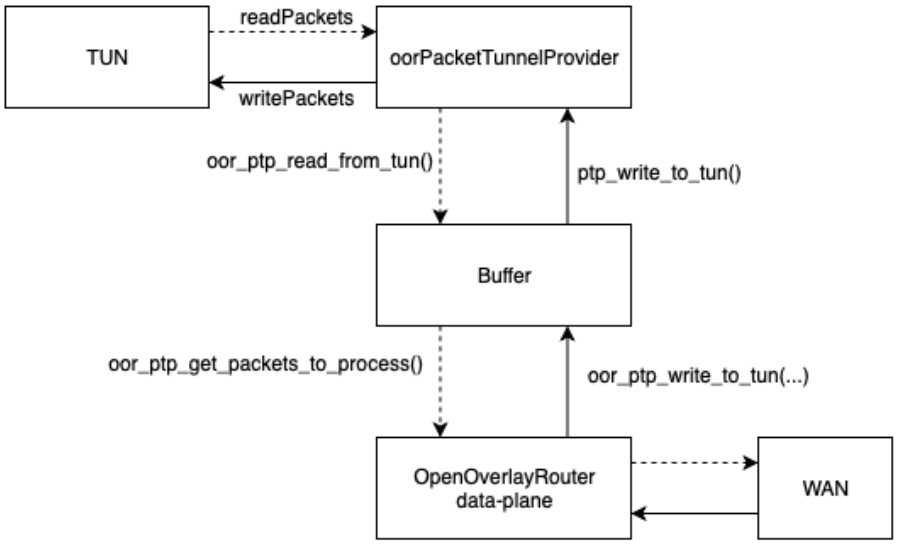
7.2. Versió 2: Intercanvi de paquets mitjançant buffers de memòria

Un cop s'ha aconseguit el funcionament de la primera versió de l'aplicació funciones, es va contactar amb enginyers d'Apple per tal de que ens ajudessin a orientar millor aquesta tasca.

Després l'intercanvi de múltiples correus, s'ha decidit realitzar una altre implementació utilitzant *buffer*s[51] de memòria i semàfors[52] per controlar l'Access a aquests, millorant així el rendiment en general de l'aplicació.

[50] https://en.wikipedia.org/wiki/Network_socket
[51] https://en.wikipedia.org/wiki/Data_buffer
[52] https://en.wikipedia.org/wiki/Semaphore_(programming)

S'ha dissenyat i implementat una API en C, *ios_packetTunnelProvider_api.h*, per tal de que tant *oorPacketTunnelProvider* com el *data-plane* de *Open Overlay Router* poguessin fer us de certs mètodes per utilitzar els *buffers*.



Il·lustració 9: Diagrama de intercanvi de paquets, segona versió

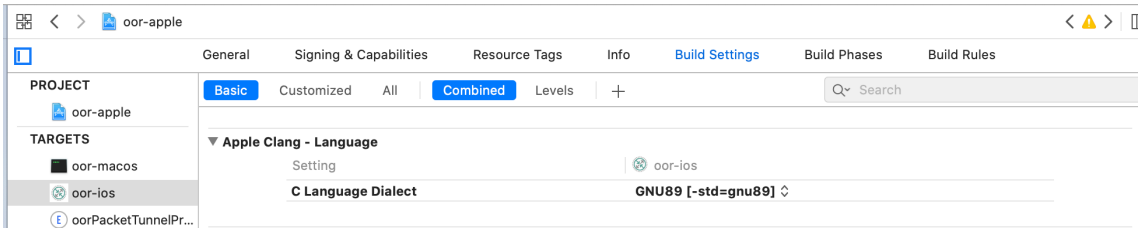
8. Desenvolupament de l'aplicació

El desenvolupament de l'aplicació s'ha distribuït en diferents fases.

8.1. Compilació amb LLVM

El primer objectiu ha sigut realitzar els canvis necessaris a l'aplicació actual, sense afegir noves funcionalitats, per tal de aconseguir que aquesta es compili amb *LLVM* per al sistema operatiu *Apple iOS* i arquitectura *ARM*[53].

Per tal de dur a terme aquesta operació s'ha hagut d'estudiar el funcionament del compilador *LLVM* i hem descobert que es necessari especificar la opció de *-std=GNU89*. Aquesta opció es pot configurar a les opcions de Xcode:



Il·lustració 10: Opció GNU89 a Xcode

8.1.1. Libconfuse

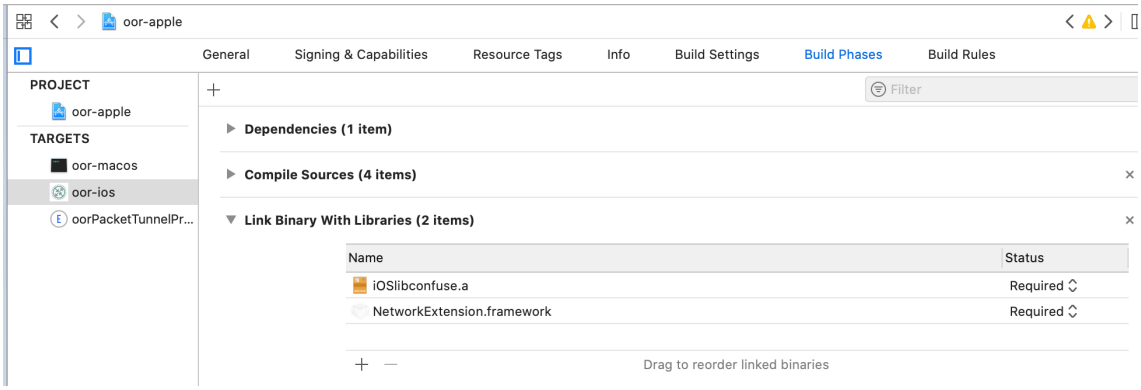
La versió de *Linux* de *Open Overlay Router* es llegeix la configuració de un fitxer amb extensió *.conf* mitjançant una llibreria anomenada *libconfuse*.

Aquesta llibreria és de codi lliure i esta disponible a un repositori de *Github*[54]. Utilitza l'eina *autoconf*[55] per tal de configurar el codi font en funció del sistema operatiu per qual es vol compilar. *Apple iOS* i l'arquitectura *ARM* no esta suportada.

[53] https://en.wikipedia.org/wiki/ARM_architecture
[54] https://github.com/martinh/libconfuse
[55] https://en.wikipedia.org/wiki/Autoconf

S'han creat els *scripts* necessaris per poder compilar *libconfuse* per al sistema operatiu *Apple iOS 10.3* i les arquitectures *arm64, armv7* i *armv7s*. Aquests scripts s'han creat a partir de uns altres scripts per a versions anteriors de *Apple iOS*. S'han publicat a *Github*[56] per tal de que siguin útils a la comunitat.

Aquests *scripts* generen un fitxer amb extensió *.a* que després es pot importar a *Xcode* com a llibreria externa.



Il·lustració 11: Importació de llibreries externes a Xcode

8.1.2. Preprocessador de C i directives de compilació

A continuació, s'ha hagut d'analitzar profundament tot el codi font en busca de llibreries que nomes existeixen a *Linux* i crides a la API de POSIX que no estan disponibles a *Apple iOS*. Per a solucionar aquest problema s'han hagut de programar varies crides del preprocessador[57] de C.

El preprocessador de C és el primer programa invocat per el compilador i processa directives com *#include, #define* o *#if*. Concretament, ens interessen les directives de compilació condicional *#ifdef, #ifndef, #else, #elif* i *#endif* conjuntament amb la macro *__APPLE__*.

[56] https://github.com/OriolOMM/iOS-autoconf-10.3)
[57] https://en.wikipedia.org/wiki/C_preprocessor

D'aquesta forma aconseguim carregar unes llibreries o unes altres en funció del sistema operatiu pel qual s'estigui compilant l'aplicació. A continuació es descriuen alguns dels usos d'aquestes directives:

- La ubicació de la llibreria *endians.h* es diferent en sistemes operatius *Linux* i *Apple iOS*:

```
23 #ifdef __APPLE__
24 #include <machine/endian.h>
25 #else
26 #include <endian.h>
27 #endif
```

Codi 1: Directives de preprocessador C endians.h
Fitxer: oor/lib/mem_util.h

- Algunes de les estructures de les capçaleres IP son diferents entre *Linux* i *Apple iOS*:

```
77 #ifdef __APPLE__
78 struct iphdr
79 {
80 #if __BYTE_ORDER == __LITTLE_ENDIAN
81     unsigned int ihl:4;
82     unsigned int version:4;
83 #elif __BYTE_ORDER == __BIG_ENDIAN
84     unsigned int version:4;
85     unsigned int ihl:4;
86 #else
87 # error  "Please fix <bits/endian.h>"
88 #endif
89     u_int8_t tos;
90     u_int16_t tot_len;
91     u_int16_t id;
92     u_int16_t frag_off;
93     u_int8_t ttl;
94     u_int8_t protocol;
95     u_int16_t check;
96     u_int32_t saddr;
97     u_int32_t daddr;
98     /*The options start here. */
99 };
100 #endif
```

Codi 2: Directives de preprocessador de C amb estructures IP
Fitxer: oor/lib/packets.h

- L'aplicació original de *Open Overlay Router* mostra els *logs* mitjançant la crida *print*. En el cas de *Apple iOS* la forma correcte és utilitzar la llibreria *syslog*.

```
#ifdef __APPLE__
        syslog(LOG_WARNING, "[%d/%d/%d %d:%d:%d] %s: ",
               tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec, log_name);
        vsyslog(LOG_WARNING,format,args);
        if (fp != NULL) {
            fprintf(fp,"[%d/%d/%d %d:%d:%d] %s: ",
                    tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec, log_name);
            vfprintf(fp,format,args);
            fprintf(fp,"\n");
            fflush(fp);
        }
#else
        printf("[%d/%d/%d %d:%d:%d] %s: ",
               tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec, log_name);
        vfprintf(stdout,format,args);
#endif
        printf("\n");
    }
#endif
}
```

Codi 3: Directives de preprocessador de C amb logs
Fitxer: oor/lib/oor_log.c

- *Netlink*[58] es una interfície del *kernel* de *Linux* pensada per a la comunicació entre processos entre el *kernel* i el espai d'usuaris. Aquesta llibreria no existeix a *Apple iOS*.

```
#ifndef __APPLE__
#include <linux/netlink.h>
#include <linux/rtnetlink.h>
#endif
```

Codi 4: Directives de preprocessador de C amb la llibreria Netlink
Fitxer: oor/lib/sockets-util.c

Com hem dit, aquestes son algunes de les més rellevants. Aconseguir que el codi compiles ha suposat la modificació de 14 fitxers de codi font afegint més de 40 directives de preprocessador de C.

Un cop s'ha aconseguit compilar, s'ha procedir a començar a dissenyar la aplicació en si, ja que ara ja tenim la garantia de que les parts clau de *Open Overlay Router* poden funcionar al sistema operatiu Apple iOS.

[58] https://en.wikipedia.org/wiki/Netlink

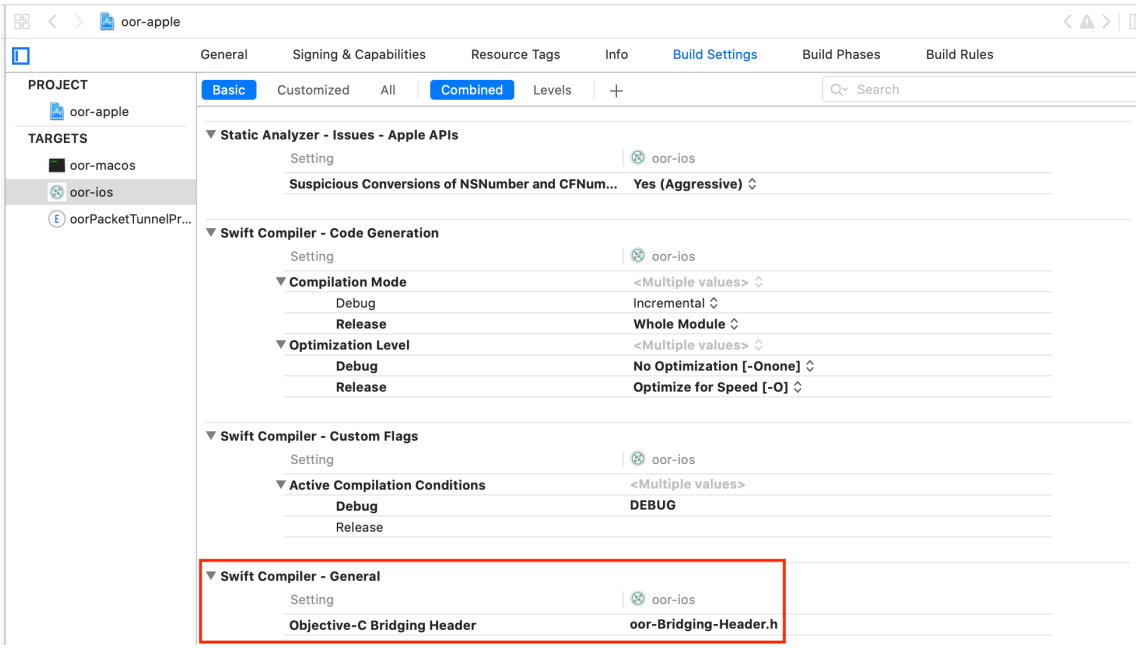
8.1.3. Bridging headers

Per tal de poder utilitzar les funcions de *Open Overlay Router* escrites en C és necessari definir el que s'anomena com a *Bridging headers*[59], la seva finalitat és exposar els mètodes d'un codi escrit en C al codi escrit en *Swift*. Això s'aconsegueix creant un fitxer capçalera *.h* de C on s'importen altres capçaleres de C que contenen els mètodes que desitgem utilitzar des de Swift.

```
9  #import "oor.h"
10 #import "ios_packetTunnelProvider_api.h"
```

Codi 5: Bridging Header
Fitxer: apple/oor-Bridging-Header.h

Finalment aquest s'ha de indicar al compilador *LLVM* la ubicació d'aquest fitxer:



Il·lustració 12: Configuració de Bridging Header a Xcode

D'aquesta forma ja podem compilar i executar el codi original de *Open Overlay Router* a *Apple iOS*.

[59] https://developer.apple.com/documentation/swift/imported_c_and_objective-c_apis/importing_objective-c_into_swift

8.2. Desenvolupament de noves parts

A continuació es descriuen les parts mes rellevants que s'han hagut de desenvolupar de nou ja que son altament dependents del sistema operatiu. Es dona especial importància al mecanisme d'intercanvi de paquets entre *oorPacketTunnelProvider* i el *data-plane* degut a que és el que ha fet que la majoria del codi sigui reutilitzable. Es descriuen les diferencies entre la primera i la segona versió de l'aplicació.

Cal remarcar, que tot el desenvolupament s'ha fet pensant en la reusabilitat del codi existent i la facilitat de manteniment a futur. S'ha valorat reescriure moltes parts directament en *Swift* però això dificultaria les tasques dels programadors futurs a l'hora d'afegir noves funcionalitats a *Open Overlay Router*. Tenint en compte això, s'ha decidit desenvolupar les mínimes parts possibles amb *Swift* i mantenir el màxim possible de funcions en la part escrita en C.

8.2.1. Paraula *Extern* i punters a funcions

Donat que *Open Overlay Router* és una aplicació que ha de funcionar sobre diferents sistemes operatius, és necessari crear una abstracció per a les crides de funcions que varien en funció del sistema operatiu sobre el que s'estigui executant.

Per a això fem us dels punters a funcions[60] de codi C i la paraula *extern*[61] per a realitzar les diferents implementacions de les funcions per cada sistema operatiu. Concretament s'utilitza per separar les implementacions del *control-data-plane*, *data-plane* i *net-mgr*.

[60] https://en.wikipedia.org/wiki/Function_pointer
[61] https://en.wikipedia.org/wiki/External_variable

Per exemple, a la capçalera *.h* del *control-data-plane* tenim el següent:

```
20 #ifndef CONTROL_DATA_PLANE_H_
21 #define CONTROL_DATA_PLANE_H_
22
23 #include "../../liblisp/liblisp.h"
24
25 typedef struct iface iface_t;
26 typedef struct uconn uconn_t;
27 typedef struct sock sock_t;
28
29 /* functions to manipulate routing */
30 typedef struct control_dplane_struct {
31     int (*control_dp_init)(oor_ctrl_t *ctrl, ...);
32     void (*control_dp_uninit)(oor_ctrl_t *ctrl);
33     int (*control_dp_add_iface_addr)(oor_ctrl_t *ctrl, iface_t *iface, int afi);
34     int (*control_dp_add_iface_gw)(oor_ctrl_t *ctrl, iface_t *iface, int afi);
35     int (*control_dp_recv_msg)(sock_t *sl);
36     int (*control_dp_send_msg)(oor_ctrl_t *ctrl, lbuf_t *buf, uconn_t *udp_conn);
37     lisp_addr_t *(*control_dp_get_default_addr)(oor_ctrl_t *ctrl, int afi);
38     int (*control_dp_updated_route)(oor_ctrl_t *ctrl, int command, iface_t *iface, lisp_addr_t *src_pref,
39             lisp_addr_t *dst_pref, lisp_addr_t *gw);
40     int (*control_dp_updated_addr)(oor_ctrl_t *ctrl, iface_t *iface,lisp_addr_t *old_addr,lisp_addr_t *new_addr);
41     int (*control_dp_update_link)(oor_ctrl_t *ctrl, iface_t *iface, int old_iface_index, int new_iface_index, int status);
42     void *control_dp_data;
43 } control_dplane_struct_t;
44
45 control_dplane_struct_t *
46 control_dp_select();
47
48 extern control_dplane_struct_t control_dp_tun;
49 extern control_dplane_struct_t control_dp_vpnapi;
50 extern control_dplane_struct_t control_dp_vpp;
51 extern control_dplane_struct_t control_dp_apple;
52
53 #endif /* CONTROL_DATA_PLANE_H_ */
```

Codi 6: Estructura control_dplane_struct
Fitxer: oor/control/control-data-plane/control-data-plane.h

Observem que la estructura *control_dplane_struct* son tot punters a funcions i a les línies 48, 49, 50 i 51 es declaren 4 variables utilitzant la paraules *extern*. Això indica al compilador que la definició d'aquestes variables es realitzarà a algun altre lloc.

Ens fixem a la línia 31 en el punter a funció *control_dp_init()* per veure com acabarà apuntant a la funció corresponent pel sistema operatiu pel qual s'està compilant l'aplicació.

Al fitxer *.c* del *control-data-plane* veiem que es defineix una funció per seleccionar una de les variables declarades anteriorment en el fitxer amb la capçalera.

```
control_dplane_struct_t *
control_dp_select()
{
#ifdef VPNAPI
    return &control_dp_vpnapi;
#elif VPP
    return &control_dp_vpp;
#elif __APPLE__
    return &control_dp_apple;
#else
    return &control_dp_tun;
#endif
}
```

Codi 7: Funció control_dp_select()
Fitxer: oor/control/control-data-plane/control-data-plane.c

Quan en alguna altre part del codi s'instancia un *control_dplane_struct* es crida a la funció *control_dp_select()* que utilitzant les directives de preprocessador retorna l'adreça corresponent a un objecte del tipus *control_dplane_struct* que esta definit a un altre fitxer.

```
control_dplane_struct_t control_dp_apple = {
    .control_dp_init = ios_control_dp_init,
    .control_dp_uninit = ios_control_dp_uninit,
    .control_dp_add_iface_addr = ios_control_dp_add_iface_addr,
    .control_dp_add_iface_gw = ios_control_dp_add_iface_gw,
    .control_dp_recv_msg = ios_control_dp_recv_msg,
    .control_dp_send_msg = ios_control_dp_send_msg,
    .control_dp_get_default_addr = ios_control_dp_get_default_addr,
    .control_dp_updated_route = ios_control_dp_updated_route,
    .control_dp_updated_addr = ios_control_dp_updated_addr,
    .control_dp_update_link = ios_control_dp_update_link,
    .control_dp_data = NULL
};
```

Codi 8: Estructura control_dp_apple
Fitxer: oor/control/control-data-plane/apple/ios/cdp_ios.c

Finalment, observem com s'associa al punter *control_dp_init* l'adreça de la funció definida al *control-data-plane* per a *Apple iOS*.

8.2.2. PacketTunnelProvider

Com ja hem comentat abans, aquest component és l'encarregat de executar el procés de *Open Overlay Router*, crear la interfície TUN, informar dels canvis en les interfícies de xarxa per al *multihoming* i reenviar els paquets d'aquesta a *Open Overlay Router* i vice-versa.

```
33      // Start fake tunnel connection
34      override func startTunnel(options: [String : NSObject]? = nil, completionHandler: @escaping (Error?) -> Void) {
35
36          self.completionHandler = completionHandler
37
38          //TUN IP address
39          let eid = defaults?.string(forKey: "eid")
40          if validateIPv4(ip: eid!) {
41              tunSettings.iPv4Settings = NEIPv4Settings(addresses: [eid!], subnetMasks: ["255.255.255.255"])
42              // Networks to be routed through TUN
43              tunSettings.iPv4Settings?.includedRoutes = [NEIPv4Route.default()]
44          } else if validateIPv6(ip: eid!) {
45              tunSettings.iPv6Settings = NEIPv6Settings(addresses: [eid!], networkPrefixLengths: [128])
46              // Networks to be routed through TUN, it appears that there is some bug with defualt IPv6 route ::/0, so we
                    define 2 routes with 2 big networks.
47              let route1 = NEIPv6Route(destinationAddress: "::", networkPrefixLength: 1)
48              let route2 = NEIPv6Route(destinationAddress: "8000::", networkPrefixLength: 1)
49              tunSettings.iPv6Settings?.includedRoutes = [route1, route2]
50          }
51
52          tunSettings.mtu = 1440
53
54          tunSettings.dnsSettings = NEDNSSettings(servers: [(defaults?.string(forKey: "dnsServer"))!])
55
56          // Apply settings and create TUN
57          setTunnelNetworkSettings(tunSettings) { error in
58              if error != nil {
59                  NSLog("PacketTunnelProvider.startTunnel.setTunnelNetworkSettingsError \(String(describing: error))")
60                  completionHandler(error)
61              }
62              // Tell to the system that the fake VPN is "up"
63              completionHandler(nil)
64          }
```

Codi 9: Funció startTunnel()
Fitxer: apple/oorPacketTunnelProvider/PacketTunnelProvider.swift

La funció *startTunnel()* és l'encarregada de configurar la interfície TUN i les rutes necessàries.

Observem a la línia 41 on es defineix com a adreça IP de la interfície TUN el *EID* assignat. A les línies 43, 47, 48 i 49 s'informa al sistema de la nova ruta per defecte per a que aquest, comenci a enviar tots els paquets cap a la interfície TUN. A destacar que pel cas de IPv6 ha sigut necessari especificar 2 grans xarxes ja que la ruta per defecte no funcionava correctament.

A la línia 52 també es defineix la MTU[62], és important definir-la ja que quan es reenvien els paquets cap al *data-plane* de *Open Overlay Router* aquest afegirà les capçaleres de *LISP* i una altre capçalera UDP.

Interessant la línia 63. Teòricament, el framework de *PacketTunnelProvider* esta dissenyat per a que el sistema no comenci a enviar paquets a la interfície TUN fins que no s'hagi establert una connexió amb un servidor VPN. En el nostre cas no existeix aquest servidor VPN ja que nomes utilitzem aquest framework perquè fins a dia d'avui és la única forma de crear una interfície TUN. Com a *workaround*, es va descobrir que posant el paràmetre a nul el sistema operatiu donava per fer que s'havia establert la connexió.

```
78      func startOOR() {
79          let fileManager = FileManager.default
80
81          let logFileSharedURL = fileManager.containerURL(forSecurityApplicationGroupIdentifier:
                "group.oor")?.appendingPathComponent("oor.log")
82
83          setLogPath((logFileSharedURL?.path as! NSString).utf8String)
84
85          let configFileSharedURL = fileManager.containerURL(forSecurityApplicationGroupIdentifier:
                "group.oor")?.appendingPathComponent("oor.conf")
86
87          setConfPath((configFileSharedURL?.path as! NSString).utf8String)
88
89          oor_start()
90
91          var endpoint: NWEndpoint
92          let packetTunnelProviderAddress = NWHostEndpoint(hostname: "127.0.0.1", port: "10001")
93
94          // Connect to OOR Data output Socket
95          endpoint = NWHostEndpoint(hostname: "127.0.0.1", port: "10000")
96          oorOut = self.createUDPSession(to: endpoint, from: packetTunnelProviderAddress)
97
98          // Start listeing incoming packets coming from OOR
99          oorOut?.setReadHandler({dataArray, error in
100             if error != nil {
101                 NSLog("PacketTunnelProvider.startOOR.oorOut.setReadhandler ERROR \(String(describing: error))")
102             }
103             self.newOORInPackets(packets: dataArray!)
104         }, maxDatagrams: 1)
105
106         oor_loop()
107     }
```

Codi 10: Funció startOOR()
Fitxer: apple/oorPacketTunnelProvider/PacketTunnelProvider.swift

La funció *startOOR()* es l'encarregada de arrancar el procés de *Open Overlay Router* i de preparar els canals de intercanvi de paquets, en el cas de la primera versió, els *sockets* i en el cas de la segona els *buffers* i semàfors.

[62] https://en.wikipedia.org/wiki/Maximum_transmission_unit

Observem la línia 89, la crida a la funció *oor_start()*. Aquesta funció esta definida dintre del codi C de *Open Overlay Router* i podem cridar-la directament gracies al *bridging header* mencionat anteriorment. Es la funció encarregada de inicialitzar el *data-plane, control-data-plane* i *net-mgr* de *Open Overlay Router*.

A la línia 96 s'estableix la connexió amb el *socket* local que ha obert el *data-plane* i s'estableix com a origen el port definit a la línia 92 per tal de que l'origen no sigui un port aleatori el *data-plane* conegui el port que te obert *oorPacketTunnelProvider* per tornar els paquets.

A la línia 99, s'inicia el procés de lectura de paquets de la interfície TUN i finalment a la 106 s'indica a *Open Overlay Router* que iniciï el processament de paquets.

Finalment a la línia 102 es on es comença a escoltar per paquets provinents del *data-plane* que han de ser tornats a la interfície TUN. En la segona versió, aquesta funció es realitza a *startTunnel()* en lloc de *startOOR()*. A continuació es mostra la crida a la funció de la API *ptp_write_to_tun()* la qual s'executarà cada cop que hi hagi paquets pendents al *buffer*.

```
43      // Start fake tunnel connection
44      override func startTunnel(options: [String : NSObject]? = nil, completionHandler: @escaping (Error?) -> Void) {
45          self.completionHandler = completionHandler
46          // Struct defined in C with the call back functions implemented in swift
47          oor_callbacks = iOS_CLibCallbacks(
48              packetTunnelProviderPtr:UnsafeRawPointer(Unmanaged.passUnretained(self.packetFlow).toOpaque()),
49              ptp_write_to_tun: {(buffer, length, afi, ptp_ptr) in
50                  autoreleasepool {
51                      unowned let myself = Unmanaged<NEPacketTunnelFlow>.fromOpaque(ptp_ptr).takeUnretainedValue()
52                      let packets = [Data(bytes: buffer, count: Int(length))]
53                      let protos: [NSNumber] = [afi as NSNumber]
54                      // Here is where the retension is produced
55                      myself.writePackets(packets,withProtocols: protos)
56                  }
57          }
58          )
```

Codi 11: Funció startTunnel(), versió 2
Fitxer: apple/oorPacketTunnelProvider/PacketTunnelProvider.swift

```
117     /// Handle outgoing packets coming from the TUN.
118     func handlePackets(_ packets: [Data], protocols: [NSNumber]) {
119         for packet in packets {
120             oorOut?.writeDatagram(packet) { error in
121                 if error != nil {
122                     NSLog("handlePackets: oorOut.writeDatagram error: \(String(describing: error))")
123                 }
124             }
125         }
126         // Read more outgoing packets coming from the TUN
127         self.packetFlow.readPackets { inPackets, inProtocols in
128             self.handlePackets(inPackets, protocols: inProtocols)
129         }
130     }
```

Codi 12: Funció handlePackets(), versió 1
Fitxer: apple/oorPacketTunnelProvider/PacketTunnelProvider.swift

A la funció *handlePackets()* que forma part de *oorPacketTunnelProvider*, ens deixa en un punter els paquets que s'han llegit de la interfície TUN i com podem veure un cop llegits s'escriuen al *socket* que tenim obert amb el *data-plane*.

```
130     /// Handle outgoing packets coming from the TUN.
131     func handlePackets(_ packets: [Data], protocols: [NSNumber]) {
132         for packet in packets {
133             let nsData = packet as NSData
134             let rawPtr = nsData.bytes
135             oor_ptp_read_from_tun(rawPtr,Int32(nsData.length))
136             oor_notify.send(string: "1")
137         }
138         // Read more outgoing packets coming from the TUN
139         self.packetFlow.readPackets { inPackets, inProtocols in
140             self.handlePackets(inPackets, protocols: inProtocols)
141         }
142     }
```

Codi 13: Funció handlePackets(), versió 2
Fitxer: apple/oorPacketTunnelProvider/PacketTunnelProvider.swift

En la segona versió de la mateixa funció observem com en comptes d'escriure en un *socket* local, s'escriu en un *buffer* a la línia 135 utilitzant una funció definida a la API desenvolupada per tal efecte.

8.2.2.1. Reachability

Una de les funcions que aporta *Open Overlay Router* és el *multihoming*. Per a poder proporcionar aquesta característica és necessari que informem a *Open Overlay Router* de l'estat de connectivitat de les interfícies de xarxa, en aquest cas concret la interfície WiFi i la LTE.

Per a això s'ha utilitzat una llibreria anomenada *Reachability*[63], la qual es proporcionada per Apple però nomes en *Objective-C*. Per tal de simplificar el projecte, s'ha agafat una versió aportada per la comunitat escrita en *Swift*.

```
168     @objc func reachabilityChanged(_ note: Notification) {
169         let reachability = note.object as! Reachability
170         let client = UDPClient(address: "127.0.0.1", port: 10002)
171 
172         if reachability.connection == .none {
173             newReachabilityStatus = 0
174             NSLog("REACHABILITY: none")
175         } else if reachability.connection == .cellular {
176             NSLog("REACHABILITY: CELLULAR")
177             newReachabilityStatus = 1
178         } else if reachability.connection == .wifi {
179             newReachabilityStatus = 2
180             NSLog("REACHABILITY: WIFI")
181         }
182 
183         if currentReachabilityStatus != newReachabilityStatus {
184             self.reasserting = true
185 
186             if newReachabilityStatus == 1 {
187                 client.send(string: "1")
188             } else if newReachabilityStatus == 2 {
189                 client.send(string: "2")
190             }
191             setTunnelNetworkSettings(tunSettings) { error in
192                 if error != nil {
193                     NSLog("PacketTunnelProvider.reachabilityChanged.setTunnelNetworkSettingsError \(String(describing:
                         error))")
194                 }
195                 // Tell to the system that the VPN is "up"
196                 self.completionHandler!(nil)
197             }
198         self.reasserting = false        }
199         client.close()
200         currentReachabilityStatus = newReachabilityStatus
201     }
```

Codi 14: Funció reachabilityChanged()
Fitxer: apple/oorPacketTunnelProvider/PacketTunnelProvider.swift

La llibreria s'utilitza a la funció *reach()* a la línia 141. Aquesta funció es l'encarregada de rebre les notificacions de *Reachability* i enviar un paquet mitjançant un altre *socket* local al *net-mgr*.

[63] https://github.com/ashleymills/Reachability.swift

Observem a la línia 170 on s'estableix la connexió amb el *socket* que ha obert el *net-mgr* quan s'ha inicialitzat i a les línies 187 i 189 l'enviament de un datagrama per notificar quina interfície te connectivitat.

8.2.3. Data-plane

El *data-plane* és l'encarregat de processar els paquets que entren i surten del dispositiu. Els hi afegeix i treu la capçalera *LISP*.

8.2.3.1. Primera versió

```
76  int
77  ios_init(oor_dev_type_e dev_type, oor_encap_t encap_type,...)
78  {
79      int (*cb_func)(sock_t *) = NULL;
80      int tun_socket, ipv4_data_socket, ipv6_data_socket;
81      int data_port;
82
83      //open socket to connect with TunnelProvider
84      tun_socket = open_data_datagram_input_socket(AF_INET, 10000);
85      sockmstr_register_read_listener(smaster, ios_output_recv, NULL,tun_socket);
```

Codi 15: Funció ios_init()
Fitxer: oor/data-plane/apple/ios/ios.c

Observem la línia 84, es on s'obre el port i comença a escoltar paquets provinents del PacketTunnelProvider.

```
172 int
173 ios_output_recv(struct sock *sl)
174 {
175     packet_tuple_t tpl;
176     lbuf_use_stack(&pkt_buf, &pkt_recv_buf, IOS_RECEIVE_SIZE);
177     lbuf_reserve(&pkt_buf, LBUF_STACK_OFFSET);
178
179     if (sock_recv(sl->fd, &pkt_buf) != GOOD) {
180         OOR_LOG(LWRN, "OUTPUT: Error while reading from tun!");
181         return (BAD);
182     }
```

Codi 16: Funció ios_output_recv()
Fitxer: oor/data-plane/apple/ios/ios_output.c

Per als paquets que venen de la interfície TUN, veiem com a la línia 179 es llegeix el *socket* local que hi ha obert entre el *oorPacketTunnelProvider* i el *data-plane*.

```
93  int
94  ios_process_input_packet(sock_t *sl)
95  {
96      uint32_t iid;
97      ios_data_t *data;
98
99      data = (ios_data_t *)dplane_apple.datap_data;
100     lbuf_use_stack(&pkt_buf, &pkt_recv_buf, MAX_IP_PKT_LEN);
101
102     if (ios_read_and_decap_pkt(sl->fd, &pkt_buf, &iid) != GOOD) {
103         return (BAD);
104     }
105
106     char *localhostIp = "127.0.0.1";
107     lisp_addr_t *tunnelProviderAddress = NULL;
108     tunnelProviderAddress = lisp_addr_new();
109     lisp_addr_ip_from_char(localhostIp, tunnelProviderAddress);
110
111     send_datagram_packet(data->tun_socket, lbuf_l3(&pkt_buf), lbuf_size(&pkt_buf), tunnelProviderAddress, 10001);
112
113     return (GOOD);
114 }
```

Codi 17: Funció ios_process_input_packet()
Fitxer: oor/data-plane/apple/ios/ios_input.c

En el cas dels paquets que entren per les interfícies de xarxa i s'han de tornar a la interfície TUN veiem com a la línia 111 s'escriuen en el *socket* local i el *oorPacketTunnelProvider* serà l'encarregat de tornar-los a la interfície TUN.

8.2.3.2. Segona versió

A la segona versió, fem us de la API creada anteriorment per tal d'accedir a un buffer compartit.

```
209 int
210 ios_output_recv2(struct sock *sl)
211 {
212     packet_tuple_t tpl;
213     glist_t * pkts_lst;
214     lbuf_t *b;
215     char buf[1024];
216     struct sockaddr_in si_other;
217     socklen_t slen = sizeof(si_other);
218     ssize_t recv_len;
219
220     pkts_lst = oor_ptp_get_packets_to_process();
221     if (!pkts_lst){
222         return (GOOD);
223     }
224
```

Codi 18: Funció ios_output_recv2()
Fitxer: oor/data-plane/apple/ios/ios_output.c

Observem la línia 220, en comptes de llegir de un *socket*, fem una crida a una funció que ens retornarà la llista de paquets.

```
94  int
95  ios_process_input_packet(sock_t *sl)
96  {
97      uint32_t iid;
98      int afi;
99
100     lbuf_use_stack(&pkt_buf, &pkt_recv_buf, MAX_IP_PKT_LEN);
101
102     if (ios_read_and_decap_pkt(sl->fd, &pkt_buf, &iid) != GOOD) {
103         return (BAD);
104     }
105     afi = pkt_afi(lbuf_l3(&pkt_buf));
106     oor_ptp_write_to_tun((char *)lbuf_l3(&pkt_buf), lbuf_size(&pkt_buf), afi);
107
108     return (GOOD);
109 }
```

Codi 19: Funció ios_process_input_packet()
Ftixer: oor/data-plane/apple/ios/ios_input.c

I el mateix succeeix per als paquets entrants, observem la línia 106 on es fa la crida per tal de tornar a escriure els paquets al *buffer* i que *oorPacketTunnelProvider* els torni a enviar a la interfície TUN.

8.2.4. Network manager (net-mgr)

El network manager és l'encarregat d'obtenir informació sobre les interfícies de xarxa així com l'adreça IP o la porta d'enllaç per defecte. A més a més és qui ordena al *control-data-plane* i al *data-plane* realitzar certes accions quan es detecta un canvi en una interfície de xarxa.

Per a rebre els canvis d'estat de les interfícies de xarxa, aquest obre un *socket* local per tal de rebre els missatges detectats per *Reachability* en el *oorPacketTunnelProvider.* Ho podem veure a la línia 101.

```
96  int ios_netm_init() {
97      int netm_socket;
98
99      //open routing socket to receive changes of network interfaces
100
101     netm_socket = socket(PF_ROUTE, SOCK_RAW, AF_UNSPEC);
102     sockmstr_register_read_listener(smaster, ios_network_changed, NULL, netm_socket);
103
104     return (GOOD);
105 }
```

Codi 20: Funció ios_netm_init()
Fitxer: oor/net_mgr/apple/ios/netm_ios.c

8.2.5. Timer Wheel

Open Overlay Router utilitza el que s'anomena un *timer wheel* per a la sincronització de totes les tasques. Les *timer wheels* son una forma de implementar cues

de *timers*, que son utilitzades per a la planificació de sucessos que han de succeir en un futur.

En la versió original de *Open Overlay Router* això esta implementat utilitzant la llibreria C *time.h*[64], que a *Apple iOS* no esta disponible degut a que no és *POSIX-certified*.

En el sistema operatiu *Apple iOS* la forma correcte de fer això és utilitzant l'anomenat *Grand Central Dispatch*[65] o també conegut com a *Dispatch*. Aquest, conte llibreries i eines del sistema que proporcionen suport a l'execució concurrent de processos.

```
48 struct timer_wheel_{
49     int num_spokes;
50     int current_spoke;
51     oor_timer_links_t *spokes;
52 #ifdef __APPLE__
53     dispatch_source_t tick_timer_id;
54 #else
55     timer_t tick_timer_id;
56 #endif
57     int running_timers;
58     int expirations;
59 } timer_wheel = {.spokes=NULL};
```

Codi 21: Estructura timer_wheel
Fitxer: oor/lib/timers.c

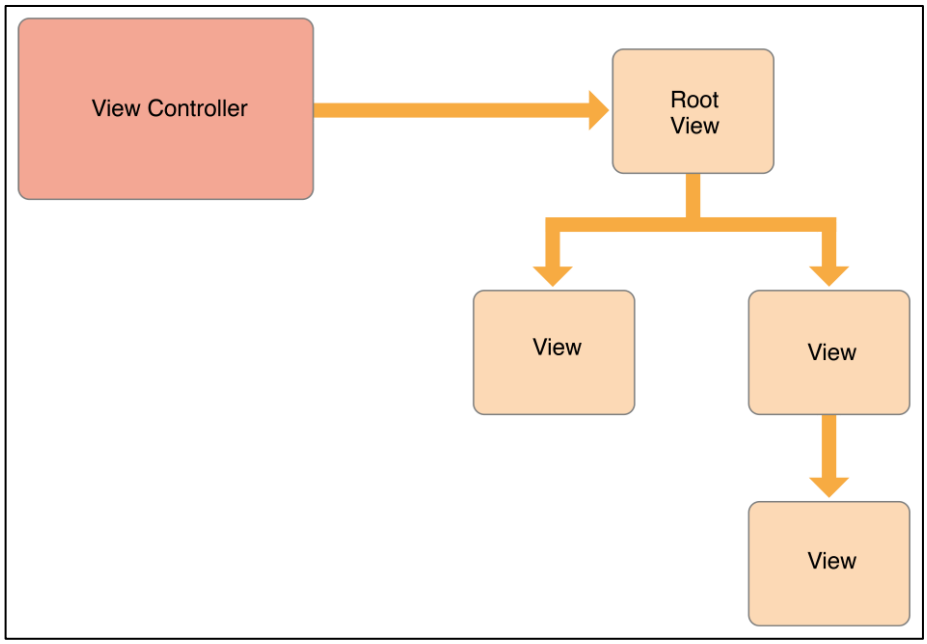
8.3. Interfície gràfica

Per a desenvolupar la interfície gràfica s'han estudiat les diferents eines visuals que aporta *Xcode* a més a més de les guies de disseny que proporciona Apple per al sistema operatiu *Apple iOS* i els patrons de disseny d'interfícies genèrics.

8.3.1. View Controllers

Els *view controllers*[66] son els fonaments de qualsevol aplicació de *Apple iOS*. Cada *view controller* gestiona una porció de la interfície gràfica de l'aplicació així com les interaccions entre la interfície gràfica i les capes de sota. A més a més, facilita les transicions entre les diferents vistes de la interfície gràfica.

[64] https://pubs.opengroup.org/onlinepubs/007908799/xsh/time.h.html
[65] https://developer.apple.com/documentation/DISPATCH
[66] https://developer.apple.com/documentation/uikit/view_controllers



Il·lustració 13: Diagrama de un view controller

L'aplicació de *Open Overlay Router* per a *Apple iOS* disposa de 3 *view controllers*. El primer orientat a controlar la vista inicial així com l'accés a les altres vistes i l'execució de oorPacketTunnelProvider. El segon per a la vista on l'usuari pot definir les opcions de l'aplicació i el tercer per a veure el registre de successos.

8.3.2. Storyboards

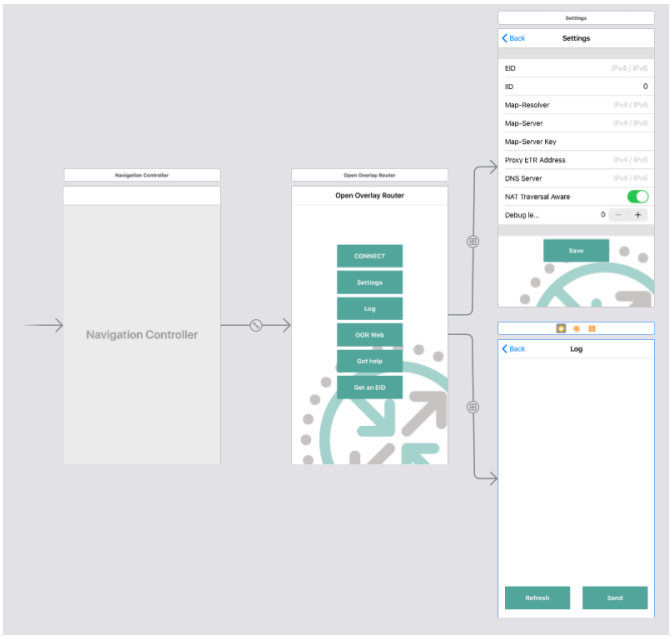
Les *storyboards*[67] es una característica introduïda a la versió 5 de *Apple iOS*. Aquesta característica estalvia temps a l'hora de desenvolupar la interfície gràfica. Ens permet prototipar i dissenyar múltiples *view controllers* en un mateix fitxer a més a més de crear les transicions entre aquests.

A continuació es mostren els dos *storyboards* que tenim a *Open Overlay Router*. El primer es exclusiu per definir la vista que es mostra quan s'obre l'aplicació.

Il·lustració 14: LaunchScreen.storyboard

[67] https://en.wikipedia.org/wiki/Storyboard#Software

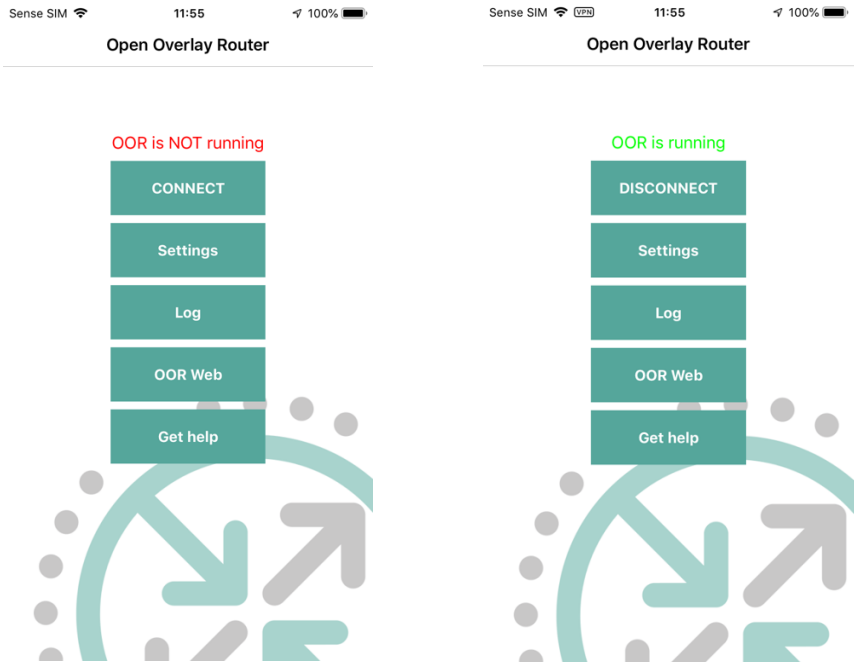
El segon, mostra totes les vistes que hi ha a l'aplicació i les transicions entre elles.



Il·lustració 15: Main.storyboard

8.3.3. Vista principal

Quan l'usuari obre l'aplicació per primer cop, la primera vista que apareixes la de configuració per tal de forçar a l'usuari introduir els paràmetres necessaris, en tots els altres casos, aquesta es la primera vista que es mostra:



Il·lustració 16: Vista principal

Des d'aquesta vista l'usuari pot saber si *Open Overlay Router* esta funcionant o no, a l'hora que també pot connectar-lo o desconnectar-lo. També podem accedir a la configuració, al registre de successos, a la web de *Open Overlay Router* i a una altre web on obtindré ajuda.

8.3.4. Configuració

Des de la vista de configuració l'usuari pot veure i editar les diferents variables que formen part de la configuració de LISP, les quals son:

- End Point Identifier (EID): Pot ser una direcció IPv4 o IPv6. Aquesta es la direcció amb la que s'identifica el dispositiu.

- Instance ID (IID): identificador per al espai de noms de adreces.

- Map-Server: Adreça IP dels servidors de *mapping* que es volen utilitzar i clau d'accés.

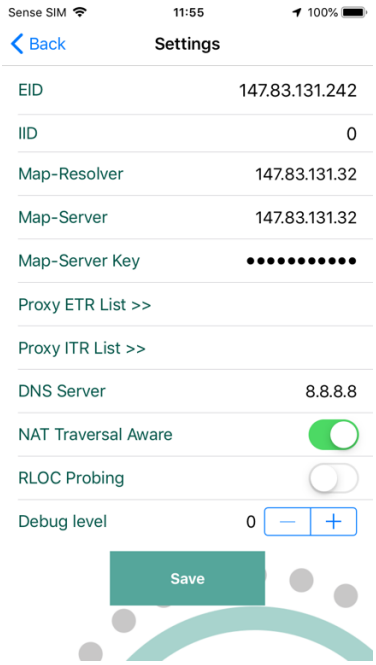
- Proxy ETR / ITR list: Llista de *proxies* ETRs i ITRs que es volen utilitzar.

- DNS Server: Adreça IP del servidor DNS.

- Nat Traversal Aware: Habilitar o deshabilitar la tecnologia de NAT Traversal

- RLOC Probing: Habilitar o deshabilitar el *probing* dels RLOC.

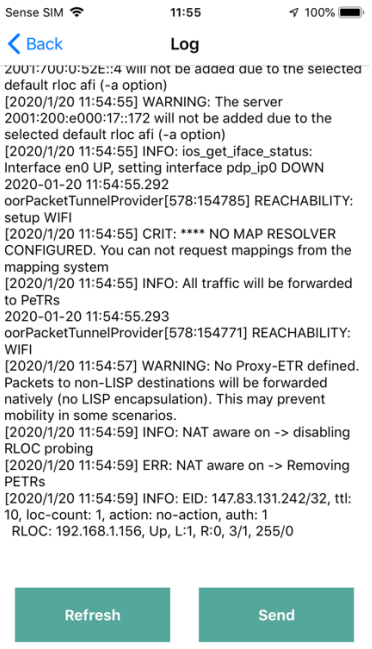
- Debug level: Del 0 al 3 el nivell de diagnosis que volem veure en el registre de successos.



Il·lustració 17: Vista de configuració

8.3.5. Registre

A la vista de registres (Log), l'usuari pot veure en detall el procés de *oorPacketTunnelProvider* i obtenir informació en cas de problemes de connectivitat. Més a més te l'opció d'enviar el registre per correu a algú per a sol·licitar ajuda en la resolució de problemes.



Il·lustració 18: Vista de registre de successos

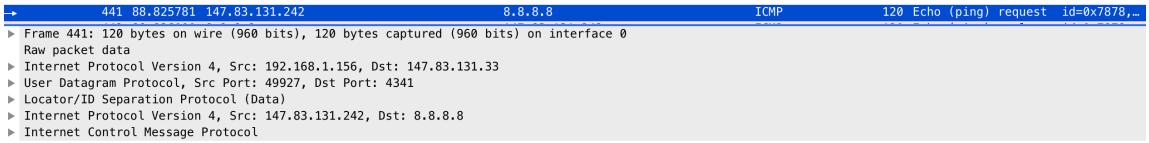
9. Resultats

En la data d'entrega de la memòria del projecte, l'aplicació *Open Overlay Router* per a *Apple iOS* es correctament funcional tant amb IPv4 com IPv6 i les funcionalitats de *LISP*, com per exemple *multihoming*, han sigut executades de forma correcte. Per tant, podem dir que *Open Overly Router* ara també es suportat pel sistema operatiu *Apple iOS*.

A continuació es mostren dues captures de *wireshark* de la interfície WiFi de un iPhone 6. Es mostra un paquet *ICMP*[68]a l'adreça IP 8.8.8.8. En la primera, no tenim *Open Overlay Router* funcionant, i per tant nomes veiem les capçaleres habituals d'un paquet *ICMP*.

Il·lustració 19: Captura de wireshark sense Open Overlay Router

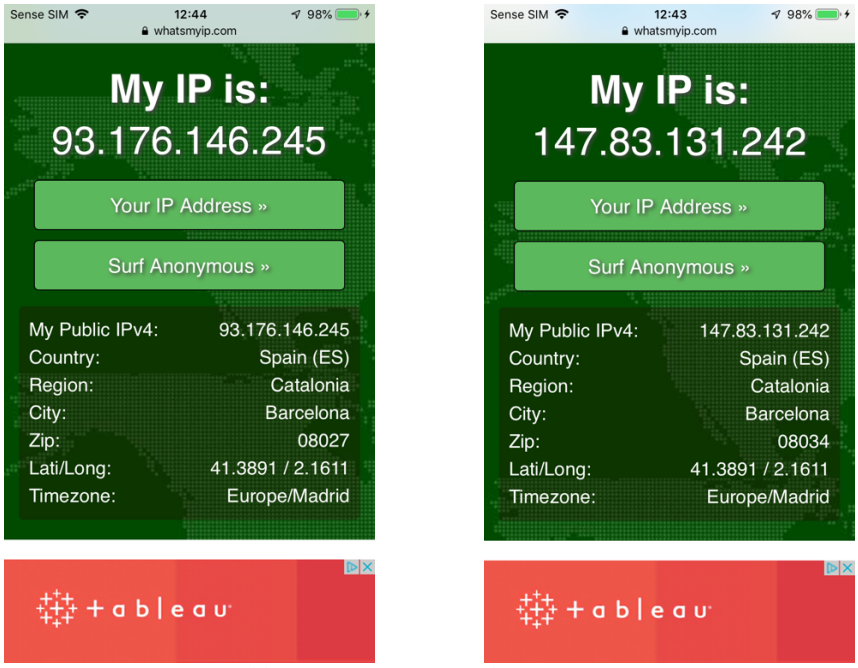
En la segona, amb *Open Overlay Router* funcionant, observem com l'origen del paquet es diferent, es a dir l'adreça IP o EID associat a la interfície TUN i com el paquet surt amb la capçalera LISP i una altre capçalera UDP.



Il·lustració 20: Captura de wireshark amb Open Overlay Router funcionant

[68] https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

Addicionalment, com a prova de funcionament, tenim 2 captures de pantalla de un iPhone 6 accedint a la web http://whatsmyip.com. A la de l'esquerra observem que ens apareix una adreça IP publica normal i a la de la dreta apareix la IP associada a la interfície TUN, es a dir el EID.



Il·lustració 21: Prova a whatsmyip.com

10. Conclusions

L'aplicació *Open Overlay Router* és de gran utilitat per a usuaris que desitgin utilitzar el protocol *LISP* a dispositius *Apple iPhone* i a més a més proporciona molta mes visibilitat al projecte *Open Overlay Router* degut a que conjuntament amb les aplicacions pels sistemes operatius *Linux*, *Android*, *OpenWrt* i ara Apple iOS s'obre la possibilitat d'arribar pràcticament a tot el mercat de dispositius i usuaris.

En quan a experiència personal, participar en aquest projecte ha sigut un gran creixement professional i m'ha aportat molts coneixements nous teòrics i pràctics de programació de sistemes operatius i comunicacions.

Mai abans havia treballat en un projecte desenvolupat en C i que tingues dependències de baix nivell amb el sistema operatiu i amb interfícies de xarxa i tampoc havia desenvolupat en el llenguatge *Swift* ni per dispositius *Apple iOS.*

Tot el conjunt del projecte ha sigut un procés de recerca i descobriment de tecnologies, patrons de programació, enginyeria de comunicacions i coneixement dels problemes actuals de les xarxes grans de Internet.

11. Treball futur

Després de l'entrega d'aquest document, l'autor es queda com a membre col·laborador del projecte *Open Overlay Router* per tal d'ajudar a incorporar progressivament les noves funcionalitats que proporcioni l'aplicació *Open Overlay Router.*

A més a més tot el projecte també s'ha desenvolupat pensant en una futura versió per a *Apple macOS* i així poder arribar a tots els dispositius del fabricant Apple.

12. Referències

[1] Apple NetworkExtension Framework, (Setembre, 2019)
https://developer.apple.com/documentation/networkextension

[2] Apple Swift, (Setembre, 2019)
https://www.apple.com/es/swift/

[3] Apple Xcode, (Setembre, 2019)
https://developer.apple.com/xcode/

[4] A. Rodríguez, M. Portoles, V. Ermagan, D. Lewis, D. Farinacci, F. Maino, A. Cabellos, Supporting Mobility in LISP.
http://www.ac.upc.edu/app/research-reports/html/2009/55/mobility lisp.pdf

[5] A. Rodríguez, What is OOR?, (Setembre, 2019)
https://github.com/OpenOverlayRouter/oor/wiki

[6] Cisco Systems, (Setembre, 2019)
https://es.wikipedia.org/wiki/Cisco_Systems

[7] D. Farinacci, The Locator/ID Separation Protocol (LISP), (Setembre, 2019)
https://tools.ietf.org/html/rfc6830

[8] LISP Mobile Node, (Setembre, 2019)
https://tools.ietf.org/html/draft-meyer-lisp-mn-06

[9] Locator/Identifier Separation Protocol, (Setembre, 2019)
https://en.wikipedia.org/wiki/Locator/Identifier_Separation_Protocol

[10] OpenOverlayRouter: Programmable Overlays, (Setembre, 2019)
http://www.openoverlayrouter.org

[11] Overview - Cisco LISP - The Locator/ID Separation Protocol, (Setembre, 2019)
http://lisp.cisco.com/lisp_over.html